



ILMENAU UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering and Information Technology

## Master Thesis

# Implementation of DSP-based Algorithms on USRP for Mitigating Non-linear Distortions in the Receiver

---

Submitted by: Florian Schlembach

Course of Studies: Communications and Signal Processing

Starting Date: 15.09.2012

Date of Submission: 14.05.2013

Department: Electronic Measurement Research Lab

Faculty of Electrical Engineering and Information Technology

Responsible Professor: Prof. Dr.-Ing. Reiner Thomä

Supervisor: Dipl.-Ing. Michael Grimm

---

# Acknowledgement

First and foremost, I owe my deepest gratitude to my supervisor Michael Grimm for the great assistance, motivation, patient guidance, encouragement and also the possibly record-breaking quick replies to all my emails. I could not have imagined having a better advisor for my master thesis.

I would like to thank Michael Huhn for his extensive technical assistance in the Mofula laboratory of the Ilmenau University of technology.

Special thanks go to Dani for her patience and her persistent encouragement.

I would like to thank my friends in Ilmenau for making the past years to an unforgettable time in my life.

My studies and this thesis would not have been possible unless the unlimited support of my family. I gratefully acknowledge their encouragement throughout my entire life.

## **Abstract**

In recent years, software-defined radio (SDR) has attracted increasingly more attention in regards to modern communication systems. The concept of SDR defines a radio device that is capable of flexibly reconfiguring its radio interface by software. This opens multiple fields of application and makes SDR an enormously adjustable and versatile radio technology.

However, RF impairments induced by cheap and simple RF front-ends turn out to be a significant limitation in practice. Non-linear distortions emerge from non-linear components of the direct down-conversion chain that are driven into their saturation level. This is a result of a finite linearity and limited dynamic range of the RF front-end.

The focus of this thesis are non-linear distortions in wideband receivers and a mitigation of them by means of digital signal processing. The idea is to artificially regenerate the non-linear distortions in the digital domain by applying a memoryless, polynomial model. An adaptive filter adjusts these reference distortions in their magnitude and phase and subtracts them from the distorted signal.

A hardware implementation of a mitigation algorithm on a typical SDR-platform is presented. No prior implementation of this pure-digital approach is known. An implementation design flow is described following a top-down approach, starting from a fixed-point high-level implementation and ending up with a low-level hardware description language implementation. Both high-level and low-level simulations help to validate and evaluate the implementation.

In conclusion, the implementation of the mitigation algorithm is a sophisticated mitigation technique for cleaning a down-converted baseband spectrum of non-linear distortions in real-time. Therefore, the effective linearity of the RF front-end is increased. This may lead to a significant improvement in the bit error rate performance of cleansed modulated signals, as well as to an enhanced sensing reliability in the context of cognitive radio.

## Kurzfassung

In den letzten Jahren sorgte Software-Defined Radio (SDR) in Bezug auf moderne Kommunikationssysteme für immer größere Aufmerksamkeit. Das Konzept von SDR bezeichnet ein Funkgerät, das in der Lage ist, seine Funkschnittstelle durch Software flexibel zu rekonfigurieren. Dies ermöglicht eine Vielzahl von Anwendungsmöglichkeiten und macht SDR zu einer enorm anpassungsfähigen und vielseitigen Funktechnologie.

Allerdings stellen im HF-Frontend ausgelöste Störungen in der Praxis eine erhebliche Einschränkung dar. In direkt umsetzenden Empfängerstrukturen entstehen durch nichtlineare Komponenten, die in ihren Sättigungsbereich getrieben werden, nichtlineare Verzerrungen. Das ist ein Ergebnis der begrenzten Linearität und des Dynamikbereichs des HF-Frontends.

Der Fokus der Arbeit liegt auf nichtlinearen Verzerrungen in breitbandigen Empfängern und deren Minderung mit Hilfe von digitaler Signalverarbeitung. Die Idee ist, die nichtlinearen Verzerrungen im digitalen Bereich auf Basis eines gedächtnislosen, Polynom-Modells zu regenerieren. Ein adaptives Filter passt dabei den Betrag der nichtlinearen Referenzverzerrungen an und subtrahiert diese vom verzerrten Signal.

In der Arbeit wird eine Hardwareimplementierung eines Störungsminderungsalgorithmus auf einer typischen SDR Plattform vorgestellt. Bisher ist keine Implementierung des rein-digitalen Ansatzes bekannt. Der Implementierungsablauf beschreibt anhand eines Top-Bottom-Ansatzes, wie der Algorithmus zuerst in einer Festpunkt High-Level Realisierung und schließlich in einer Low-Level Implementierung mit einer Hardwarebeschreibungssprache umgesetzt wird. Sowohl High-Level als auch Low-Level Simulationen unterstützen dabei die Validierung und Bewertung der Implementierung.

Die Implementierung des Abschwächungsalgorithmus stellt schließlich eine ausgefeilte Methode dar, um ein heruntergeschmisches Basisbandspektrum in Echtzeit von nichtlinearen Verzerrungen zu befreien. Demzufolge wird die effektive Linearität des HF-Frontends erhöht. Dies kann zu einer erheblichen Verbesserung der Bitfehlerrate von modulierten Signalen führen sowie die Zuverlässigkeit des Sensings in Bezug auf kognitiven Funk steigern.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Problem Statement . . . . .	2
1.2	State of the Art . . . . .	3
1.3	Goal of the Thesis . . . . .	4
1.4	Outline of the Thesis . . . . .	4
<b>2</b>	<b>Fundamentals</b>	<b>5</b>
2.1	Direct-conversion Receiver . . . . .	5
2.1.1	I/Q Signal Processing . . . . .	6
2.1.2	RF Impairments . . . . .	8
2.2	Intermodulation Distortions . . . . .	9
2.3	FIR Filter . . . . .	12
2.4	LMS Adaptive Filter . . . . .	15
2.4.1	Interference Cancellation . . . . .	16
2.4.2	Widrow-Hoff LMS Filter . . . . .	17
2.5	Fixed-point Arithmetic . . . . .	19
2.5.1	Binary Number Representations . . . . .	19
2.5.2	Bit Growth Considerations . . . . .	20
2.5.3	Rounding Methods . . . . .	21
2.6	Field Programmable Gate Array . . . . .	21
2.6.1	Architecture . . . . .	22
2.6.2	Design Flow . . . . .	24
<b>3</b>	<b>The NONLIM Algorithm</b>	<b>28</b>
3.1	Reference Branch . . . . .	29
3.2	Desired Branch . . . . .	30
3.3	LMS Adaptive Filter . . . . .	31
3.4	Simplifications and Limitations . . . . .	32

<b>4</b>	<b>Implementation</b>	<b>33</b>
4.1	Target Architecture . . . . .	33
4.1.1	DSP48A Slice . . . . .	33
4.1.2	Available Resources . . . . .	34
4.1.3	Analogue-to-digital Converter . . . . .	34
4.1.4	RF Front-end . . . . .	35
4.1.5	Analysis of the Existing DDC Chain . . . . .	35
4.2	Implementation Design Flow . . . . .	37
4.2.1	Requirement Analysis . . . . .	39
4.2.2	Experimental MATLAB Script . . . . .	39
4.2.3	Migration to HDL Module . . . . .	40
4.2.4	HDL Testbench . . . . .	40
4.2.5	MATLAB-HDL Interface SigIO . . . . .	43
4.2.6	Synthesis and Implementation . . . . .	45
4.2.7	FPGA Programming . . . . .	45
4.3	Modules Implementation . . . . .	46
4.3.1	Implementation Constraints . . . . .	46
4.3.2	Integration into Existing Firmware . . . . .	46
4.3.3	FIR Filters . . . . .	48
4.3.4	Reference Model . . . . .	54
4.3.5	Delay Blocks . . . . .	55
4.3.6	LMS Adaptive Filter . . . . .	57
4.3.7	Resource Utilisation . . . . .	66
4.4	Processing Delay of NONLIM . . . . .	68
4.5	Complex-valued LMS Implementation . . . . .	68
<b>5</b>	<b>Measurement Results</b>	<b>71</b>
5.1	Two-tone Scenario . . . . .	71
5.2	BPSK Scenario . . . . .	73
5.3	Discussion . . . . .	75
<b>6</b>	<b>Future Work</b>	<b>77</b>
6.1	Complex-valued LMS Filter . . . . .	77
6.2	Extensions to Arbitrary Interferer Signals . . . . .	77
6.3	Runtime Reconfigurability . . . . .	79
<b>7</b>	<b>Conclusion</b>	<b>81</b>

---

<b>A Experimental Setup</b>	<b>88</b>
<b>Bibliography</b>	<b>89</b>
<b>List of Figures</b>	<b>94</b>
<b>List of Tables</b>	<b>96</b>
<b>Theses</b>	<b>97</b>

# 1 Introduction

In recent years, software-defined radio (SDR) radio has increasingly attracted more attention in regards to modern communication systems. Usually, conventional communication systems reside on a hardwired printed circuit board (PCB), which are configured to operate according to a specific communication protocol standard. They comprise typical defined radio parameters such as the carrier frequency, a certain bandwidth and a modulation scheme. For instance, a modern smartphone contains multiple radio interfaces. Each of them is dedicated to a special purpose such as establishing phone calls within the mobile cell, receiving the current position via the Global Positioning System (GPS) system, operating within a WiFi network or even paying the public transport ticket based on the radio-frequency identification (RFID) technology [1].

In contrast, a SDR terminal is not designed for a specific application but is capable of seamlessly reconfiguring the key parameters of the radio interface. This allows the interaction of the SDR device with any radio environment. As implied by the name, the entire functionality of the SDR-platform can be reconfigured by software, which opens multiple fields of application and makes the SDR terminal an enormously flexible and versatile radio device [2]. This is enabled by the fact that most of the transceiver functionalities are pushed towards the digital domain, whereas only a few processing stages, such as amplification and up-/down-conversion, are still remained in the analogue domain. The conversion between the analogue and digital domain is established by analogue-to-digital converters (ADCs) and digital-to-analogue converters (DACs). Thereafter, high-speed processing operations, such as interpolation and decimation of high sample rates, are usually performed by a field programmable gate array (FPGA). The digitised and decimated samples of the received radio signal are then sent to a host PC via an Universal Serial Bus (USB) or a gigabit ethernet (GigE) interface, where further signal processing is carried out by a general-purpose processor (GPP). Hence, the type of application of the radio terminal is freely configured by the piece of software running on the personal computer (PC). Additionally, high-speed processing steps can be displaced to the FPGA due to its inherent flexibility, as they are highly



programmable digital circuits. This makes the performance of computationally intensive operations in real-time possible what allows for a flexible and rapid implementation and evaluation of novel signal processing algorithms in realistic radio environments [3].

However, most of the SDR platforms are aimed to be simple and cheap. Therefore, they are using a homodyne receiver, or direct-conversion receiver (DCR), architecture [3]. Hereby, radio frequency (RF) impairments are implied, what leads to the problem statement of the thesis.

## 1.1 Problem Statement

RF impairments become apparent as new unwanted frequency components, phase noise, jitter and in-phase/quadrature (I/Q) imbalance effects within the down-converted baseband (BB), which degrade the RF performance and cause a finite linearity and limited dynamic range of the RF front-end [4, 5]. They mainly originate from strong signals driving the RF components of the DCR chain into their non-linear ranges. These type of distortions are thus referred to as non-linear distortions being a subset of RF impairments. The focus of this thesis is on the non-linear distortions only.

In recent years, the maximum supported bandwidth of SDR devices has been increasing, as the technology of FPGA is highly evolving and A/D-converters are supporting increasingly fast conversion rates. As a consequence, the problem of non-linear distortions of wideband SDR receivers becomes even more significant as both strong and weak signals are received simultaneously [3]. Strong signals might cause severe distortions, which are spread over the entire received BB spectrum.

Scenarios that are especially prone to distortion effects use the frequency-division multiplexing (FDM) technique, where the whole frequency bandwidth is divided into frequency sub-bands, or channels. In practice, the affected real-world scenarios can be briefly divided into a neighbouring frequency-division multiple access (FDMA)-channels scenario, where a strong neighbouring frequency channel induces out-of-band distortions, and a cognitive radio (CR) sensing scenario, whereby the sensing reliability is significantly deteriorated by non-linear distortions [3].

Regarding the neighbouring FDMA-channel scenario, different frequency channels are allocated to different users so that they only transmit and receive on this specific frequency range. The use case where strong frequency channels create non-linear out-of-band distortions, which fall into the target channel, does not seem to be critical if both channels belong to the same mobile network. In this case, a power control mechanism employed by most of the FDM communication standards, such as Global

System for Mobile Communications (GSM), avoids these co-channel interferences [6]. However, if the neighbouring channel does not belong to the same mobile network or is not even of the same communication standard, non-linear distortions cause severe interference, which could result in a deteriorated bit error rate (BER) performance [7].

The concept of CR classifies the users as primary users (PUs) and secondary users (SUs) [8]. The PU purchases the spectrum license and has the primary right to use it. Besides, SUs are allowed to use unoccupied ranges within the spectrum in order to increase the spectral efficiency. Thereby, the SU senses the received wideband spectrum for one of these spectrum holes. However, if PUs are communicating over a certain frequency channel with a high power, a spectrum hole, which could be detected, is likely to be affected by non-linear distortions. Thus, the spectrum hole could be sensed as occupied and a transmission opportunity would be missed for the SU. Hence, non-linear distortions would degrade the sensing reliability within a CR radio environment.

## 1.2 State of the Art

A system-level approach to mitigate the non-linear distortions by means of digital signal processing was firstly presented in [4]. Basically, the regeneration of the non-linear distortions takes place in the digital domain, which is based on a memoryless, polynomial model. The regenerated distortions are adjusted and subtracted from the actual distortions using an adaptive filter (AF). The same algorithm was further investigated by [3] in terms of increasing the reliability of spectrum sensing. [9] pursues, in turn, a mixed-signal approach of the same mitigation algorithm. Hereby, the non-linear distortions are regenerated in a custom designed analogue RF front-end of an Universal Mobile Telecommunications System (UMTS) receiver. Only the cancellation of the non-linear distortions is performed in the digital domain using an FPGA. Likewise, this approach represents the only prior implementation of this algorithm. Hereby, certain practical limitations of the pure-digital approach of [4, 3] are overcome by regenerating the non-linear distortions at RF level, thus including the entire bandwidth of the RF front-end. General considerations about FPGA implementations of an AF are discussed in [10]. In [11], an optimisation of a least mean square (LMS) AF implementation is considered. Thereby, the computation of the adaptation process is simplified by a simple bit-shift operation, which represents a more resource efficient FPGA realisation.

However, there is no implementation of the pure-digital approach on a typical commercially available SDR-platform known so far. Although the mixed-signal approach

of [9] offers several advantages, it constitutes likewise a very complex solution since a highly customised RF front-end is required. In contrast, the purely-digital approach of [4, 3] can be easily implemented on the FPGA, as today's FPGA devices offer plenty of available resources. Since no modification of the RF front-end is required, this approach suits perfectly the concept of SDR.

## 1.3 Goal of the Thesis

The main scope of this thesis is to present a hardware implementation of the system-level approach, which was firstly introduced in [4]. The non-linearly induced interference mitigation (NONLIM) algorithm cancels the non-linear distortions by applying an adaptive feed-forward mitigation algorithm by means of digital signal processing. The implementation is to be carried out on an FPGA, Xilinx Spartan-3A DSP, which is employed on the SDR-platform Universal Software Radio Peripheral (USRP) N210 [12]. Eventually, the correct functionality is to be verified by adequate simulations and measurements.

## 1.4 Outline of the Thesis

The thesis is organised as follows. In Chapter 2, fundamentals about filter theory, number representations and the FPGA architecture are discussed. Thereafter, the implementation of the NONLIM algorithm is explained in detail in Chapter 3. Then, an implementation design flow is introduced in Chapter 4, demonstrating the approach of the implementation of each individual sub-module of NONLIM. Thereby, high-level (HL) MATLAB and hardware description language (HDL) simulations validate the correct functionality of the sub-modules. In Chapter 5, measurement results are presented in order to verify the successfully implemented NONLIM algorithm. Finally, this thesis will be concluded in Chapter 6, giving an outlook on future work.

## 2 Fundamentals

### 2.1 Direct-conversion Receiver

The heterodyne and the homodyne concepts are the two main receiver architectures being used in modern communication systems. The heterodyne architecture has proven to exhibit an excellent performance in terms of selectivity of a certain frequency band and sensitivity for receiving a minimal signal strength with an acceptable signal-to-noise ratio (SNR) [13]. However, heterodyne receivers incorporate a higher complexity, are large in form-factor, expensive by means of their production costs, and contain more RF components due to additional intermediate frequency (IF) mixing stages for enhancing the selectivity. In contrast, most of the SDR devices set the design goal of being low-cost, dissipating low power and being small in their form-factor [13]. Thus, the heterodyne approach is fairly unsuitable for use in SDR transceivers, whereas the homodyne architecture (also referred to as the DCR architecture) is most commonly used for SDR devices.

Homodyne receivers reduce the down-conversion chain to a basic circuitry and are simpler, cheaper and more compact. Compared to the heterodyne concept, the homodyne architecture is simplified by skipping the intermediate stage of mixing the RF signal to an IF [13]. Instead, the DCR architecture converts the RF signal directly to BB, whereby it is also denoted as a direct down-conversion (DDC) receiver architecture. A main disadvantage of the homodyne concept is that there are several RF impairments such as DC offset, I/Q imbalance and non-linear distortions implied, as discussed later in Section 2.1.2. One type of non-linear distortions are intermodulation distortions (IMDs), which are the main scope of this work. The implementation target USRP N210 will be the subject of this thesis [14]. Therefore, the focus is based on the DCR architecture.

Figure 2.1 illustrates the DCR chain of an SDR receiver. The purpose of a CR interface is that it should be capable of receiving a multi-mode or multi-band signal. Therefore, the receiver chain consists of a wideband RF front-end that firstly pre-amplifies an incoming RF passband signal  $x_{\text{RF}}(t)$  with a low-noise amplifier (LNA)

and then converts it directly down to a complex BB signal  $\tilde{x}(t)$ .

A passband and a BB signal are defined at which carrier frequency most of the signal energy is located at a certain carrier frequency  $f_c$  and around DC, respectively [15]. Thereby, the centre frequency of the passband is determined by the local oscillator (LO). After the mixing stage, high frequency components will be removed using a low-pass filter with a cut-off frequency of around  $f_s/2$ . The BB signal  $\tilde{x}(t)$  is then amplified with a baseband amplifier (BB-A) and ready to enter the digital back-end. After it has been digitised by an ADC, further digital signal processing (DSP) takes place in an FPGA and a PC, which acts as the host of the SDR interface. In general, the expensive DSP operations (e.g. decimation of a high input sample rate) mostly remain on the FPGA. The remainder of the signal processing (e.g. demodulation) is being performed on the PC that provides the tremendous benefit of flexibility to tune the parameters of the radio interface at runtime.

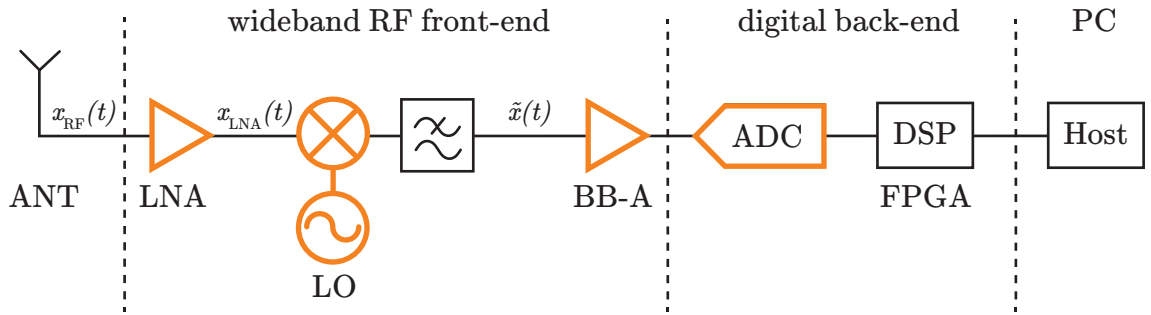


Figure 2.1: DCR architecture with the non-ideal components highlighted in orange, taken from [3]

### 2.1.1 I/Q Signal Processing

The principle of in-phase/quadrature (I/Q) forms the basis for any spectrally efficient modulation and demodulation scheme, which are commonly used in communication systems. Thereby, the signal bears the information in its amplitude and phase. It is used for efficiently up- and down-converting a passband signal to BB and vice versa. In general, a real-valued signal  $x_{\text{RF}}(t)$  can be represented as

$$\frac{1}{2}(x_{\text{RF}}(t) + j\mathcal{H}\{x_{\text{RF}}(t)\}), \quad (2.1)$$

where  $\mathcal{H}\{x_{\text{RF}}(t)\}$  denotes the Hilbert transform [15]. The spectrum shifting operation is then accomplished by multiplying (2.1) by  $e^{-j\omega_c t}$ , yielding the so-called *complex*

envelope

$$\tilde{x}(t) = \frac{1}{2}(x_{\text{RF}}(t) + j\mathcal{H}\{x_{\text{RF}}(t)\})e^{-j\omega_c t}, \quad (2.2)$$

where  $\omega_c$  equals  $2\pi f_c$ . The *complex envelope* defines the complex-valued counterpart  $\tilde{x}(t)$  of a real-valued signal  $x_{\text{RF}}(t)$  with respect to  $f_c$  [15]. Hence, the operation defined in (2.2) shifts the spectrum that is originally concentrated at  $f_c$  to DC, forming an odd, complex-valued spectrum. The real-valued signal can be recovered by

$$x_{\text{RF}}(t) = 2 \operatorname{Re}\{\tilde{x}(t)e^{j\omega_c t}\}. \quad (2.3)$$

In practice, it is common to consider the real and imaginary component of the *complex envelope*  $\tilde{x}(t)$  as two separated, real-valued, parallel signals  $x_I(t) = \operatorname{Re}\{\tilde{x}(t)\}$  and  $x_Q(t) = \operatorname{Im}\{\tilde{x}(t)\}$ . Both are referred to as the *inphase*- and *quadrature component*, respectively. Expanding (2.2) by inserting (2.3) yields the real-valued BB representation

$$x_{\text{RF}}(t) = 2x_I(t) \cos(\omega_c t) - 2x_Q(t) \sin(\omega_c t), \quad (2.4)$$

where  $x_I$  and  $x_Q$  can easily be further processed in the BB [13, 5]. Likewise,  $x_{\text{RF}}(t)$  could be retrieved from  $\tilde{x}(t)$  by

$$x_{\text{RF}}(t) = \tilde{x}(t)e^{j\omega_c t} + \tilde{x}^*(t)e^{-j\omega_c t}, \quad (2.5)$$

where the *complex envelope*  $\tilde{x}(t)$  and its complex-conjugated version  $\tilde{x}^*(t)$  is shifted to  $\omega_c$  and  $-\omega_c$ , respectively. Practically, the down-converter can be implemented as depicted on the left-hand side of Figure 2.2. As a result, the incoming real-valued signal  $x_{\text{RF}}(t)$  is down-converted to BB by the multiplication of two phase-shifted sinusoids. The equivalent representation of the *complex envelope* is shown on the right-hand side of Figure 2.2. Furthermore, the mixing operation causes additional high frequency components that are subsequently filtered out using a low-pass filter [13].

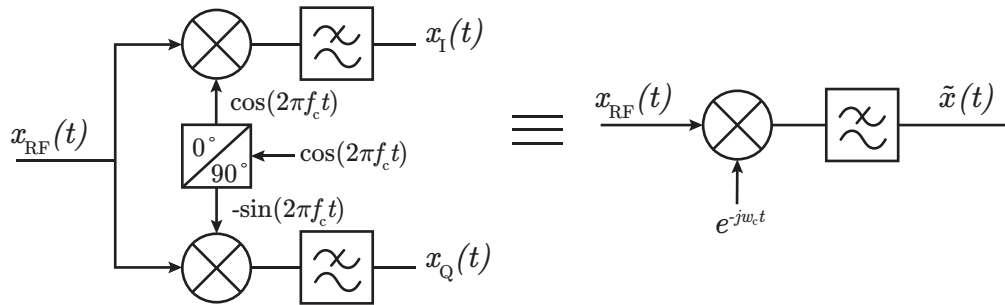


Figure 2.2: I/Q down-converter using two phase-shifted sinusoids, taken from [15]

### 2.1.2 RF Impairments

Most of the SDRs devices are aimed to be low-cost and compact, and should dissipate only low-power. In order to meet these goals, they usually accommodate cheap direct-conversion receiver (DCR) front-ends. One drawback of these DCR front-ends is that RF impairments are caused by analogue components, as highlighted in orange in Figure 2.1. The main RF impairments are summarised in the following:

**DC Offset** One of the more serious problems that originates from the weak isolation between the LO and the LNA. Thereby, a capacitive and substrate coupling causes an LO leakage and results in a self-mixing of the LO signal with itself. As a result, a DC component appears at the output of the mixers what could corrupt any modulated signal [13]. There are several methods like an AC-coupling filter that can be applied in order to get rid of the DC offset [16]. However, handling of DC offset is not within the scope of this thesis.

**I/Q imbalance** Another impairment is the I/Q imbalance. Although they arise from different non-linear components in the DDC chain, their main cause is the mixer, which incorporates both an amplitude and phase mismatch between the I/Q components. Similar to the DC offset, I/Q imbalance will be present in further proceedings but is not within the primary scope of this thesis. It is referred to in the literature [4], [13] and [5].

**Non-linear distortions** Furthermore, non-linear distortions arise from the non-linear components of the receiver architecture. These are usually amplifiers and mixers, however, amplifiers are mostly the main source for non-linear distortions [5]. Their non-linear behaviour can basically be described by the input-output characteristic of the amplifier, as shown in the following section. Once their input level is driven towards the saturation level of the amplifier, spurious distortions are generated. In terms of the ideal spectrum of the input signal, new frequency components appear at the output. It can be distinguished between two types of non-linearities: Harmonics and intermodulation distortions (IMDs). Harmonics constitute multiple integers of the input frequencies, whereas IMDs describe linear combinations of the input frequencies [5]. Details of the former can be found in the literature [17]. The latter are the main focus of this thesis and will be discussed in the following.

## 2.2 Intermodulation Distortions

Intermodulation distortions (IMDs) can briefly be divided into two main types of in-band and out-of-band distortions, depending on the frequency bands to be considered. In-band distortions emerge within the target frequency band and become apparent in terms of an increased noise floor of the target band [5]. Out-of-band distortions, in contrast, originate from interfering and blocking signals out of the desired target band. If they appear close to the spectrum band they are originating from, they are called spectral regrowth. The issue of out-of-band distortions is even more problematic if a wideband spectrum is received from a multi-band SDR interface and no pre-selection filter is applied before the LNA [3]. Hereby, the wideband spectrum is down-converted with both strong out-of-band blocking signals and the desired target signal band [3]. The strong blocking interferer causes non-linear, spurious distortions that can easily hit the target band. This might result in a corrupted demodulation and thus a deteriorated BER performance of the received signal. Another situation, where non-linear distortions lead to poor performance, occurs when a CR device senses its radio environment. Hereby, the mobile terminal tries to make use of unused spectrum holes. A spectrum band affected by non-linear distortions could be sensed as occupied and a transmission opportunity would be missed.

An approximation of the non-linear distortions of a non-linear component can be described by a polynomial, memoryless model

$$y(t) = \sum_{n=1}^{\infty} a_n x^n(t) = a_1 x(t) + a_2 x^2(t) + a_3 x^3(t) + \dots, \quad (2.6)$$

where  $a_i$  with  $i = 1, 2, 3, \dots$  denotes the real-valued, polynomial coefficients, whereas  $x(t)$  and  $y(t)$  correspond to any input and output signal of the non-linear component, respectively. Since the polynomial coefficients are assumed to be real-valued, the model takes only amplitude-to-amplitude (AM/AM) distortions into account. If the model is extended by complex-valued coefficients  $a_i$ , an amplitude-to-phase (AM/PM) relation could be established. The non-linearity described by (2.6) is composed of the sum of the original signal multiplied with the linear-gain  $a_1$  plus the weighted even- and odd-order distortion products. According to [3], third-order products created by the LNA seem to be the most significant sources of non-linear distortions using the WBX front-end [12]. Thus, the focus of this thesis is based on the non-linear behaviour of the LNA only. Consequently, (2.6) reduces to the linear and the third-order term,



becoming

$$x_{\text{LNA}}(t) = a_1 x_{\text{RF}}(t) + a_3 x_{\text{RF}}^3(t). \quad (2.7)$$

In Figure 2.3, the non-linear input-output characteristic is shown, choosing the polynomial coefficients  $a_1 = 1$ ,  $a_3 = -0.2$ . The figure demonstrates the deviation between the ideal, linear and the approximated non-ideal, non-linear amplifier characteristic. The higher the magnitude of the input voltage the more the output voltage becomes compressed. Hence, the sign of the third-order coefficient  $a_3$  is always negative. This characteristic causes various, additional frequency components, as analysed below.

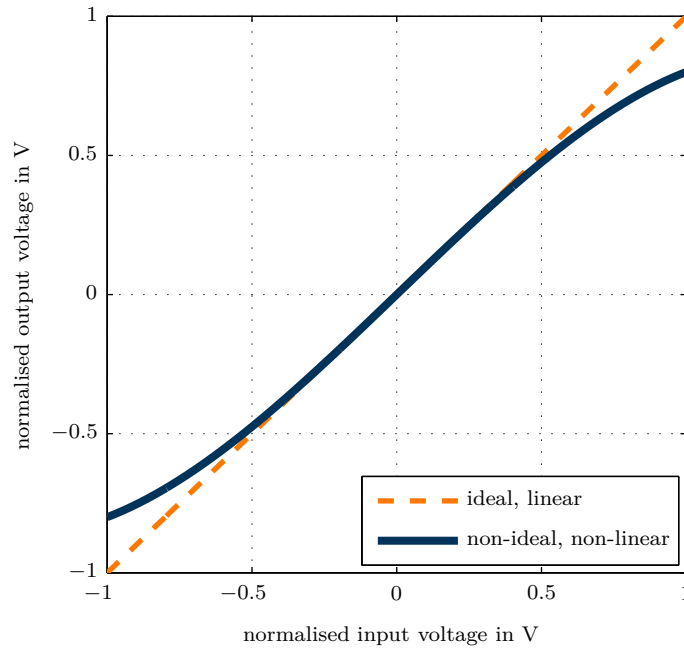


Figure 2.3: Approximated non-linear input-output characteristic of an amplifier applying the memoryless, polynomial model

For this purpose, it is now investigated how the RF input signal  $x_{\text{RF}}(t)$  passes through the RF front-end of the DCR chain of Figure 2.1 and ends up being down-converted to the complex BB signal  $\tilde{x}(t)$ . First,  $x_{\text{RF}}(t)$  is amplified by the LNA, which is by assumption the dominant non-linear component within the receiver chain. Hereby, the non-linearity of (2.7) is applied by substituting (2.5) into (2.7). After some straightforward rearrangements (the time variable  $t$  is skipped for sake of brevity) it

can be obtained

$$\begin{aligned}
 x_{\text{LNA}} &= a_1(\tilde{x}e^{j\omega_c t} + \tilde{x}^*e^{-j\omega_c t}) + a_3(\tilde{x}e^{j\omega_c t} + \tilde{x}^*e^{-j\omega_c t})^3 = \dots \\
 &= \underbrace{(a_1\tilde{x} + 3a_3\tilde{x}^2\tilde{x}^*)}_{\text{left over in BB}}e^{j\omega_c t} + (a_1\tilde{x}^* + 3a_3\tilde{x}\tilde{x}^{*2})e^{-j\omega_c t} + a_3\tilde{x}^3e^{3j\omega_c t} + a_3\tilde{x}^{*3}e^{-3j\omega_c t},
 \end{aligned} \tag{2.8}$$

which already reveals the harmonics of  $\omega_c$  generated by the non-linearity of the LNA. According to Figure 2.1, the resulting signal  $x_{\text{LNA}}$  is now multiplied by  $e^{-j\omega_c t}$ , which then left shifts the entire spectrum by  $f_c$ . The harmonics of  $\omega_c$  are most likely filtered out by a low-pass filter (with a bandwidth (BW) of approximately  $2f_s$ ) and the distorted complex BB signal becomes

$$\tilde{x}_{\text{dist}}(t) = (a_1\tilde{x} + 3a_3\tilde{x}^2\tilde{x}^*). \tag{2.9}$$

For a further analyses of the distorted BB spectrum of signal  $\tilde{x}_{\text{dist}}(t)$ , a test signal with a known spectrum is crucial. Thus, a two-tone test signal was chosen, which is composed of two complex sinusoids with a certain frequency spacing, yielding

$$x_{\text{tt}}(t) = e^{j2\pi f_0 t} + e^{j2\pi f_1 t}, \tag{2.10}$$

where  $f_0$  and  $f_1$  are the frequencies of the two tones that correspond to two unit pulses in the BB spectrum. The two-tone signal  $x_{\text{tt}}(t)$  is now fed into (2.9) as  $\tilde{x}(t)$ , forming the distorted complex BB spectrum

$$\begin{aligned}
 \tilde{x}_{\text{dist}} &= a_1\tilde{x} + a_3\tilde{x}^2\tilde{x}^* \\
 &= a_1(e^{j\omega_0 t} + e^{j\omega_1 t}) + a_3(e^{j\omega_0 t} + e^{j\omega_1 t})^2(e^{j\omega_0 t} + e^{j\omega_1 t})^* = \dots \\
 &= (a_1 + 3a_3)e^{j\omega_0 t} + (a_1 + 3a_3)e^{j\omega_1 t} + \underbrace{a_3e^{j(2\omega_0 - \omega_1)t} + a_3e^{j(2\omega_1 - \omega_0)t}}_{\text{additional frequency components}}.
 \end{aligned} \tag{2.11}$$

Hence, it is apparent that the distorted BB spectrum now consists of two additional unit pulses located at  $2f_0 - f_1$  and  $2f_1 - f_0$ , which are, in the following, referred to as the third-order intermodulation (IM3) products. Likewise, it also shows in-band distortions as additional components appear at the original frequency positions  $f_0$  and  $f_1$ , which are weighted with a factor  $3a_3$  [4]. In order to illustrate the appearance of these components, Figure 2.4 depicts a simulated power spectral density (PSD) of the distorted signal  $\tilde{x}_{\text{dist}}$ . The two tones are located at 1.2 MHz and 1.8 MHz with a power of  $-20$  dBm and the noise floor being at approximately  $-75$  dBm. Hence, the

IM3 products appear at 0.6 MHz and 2.4 MHz, in accordance with the positions being evaluated by (2.11).

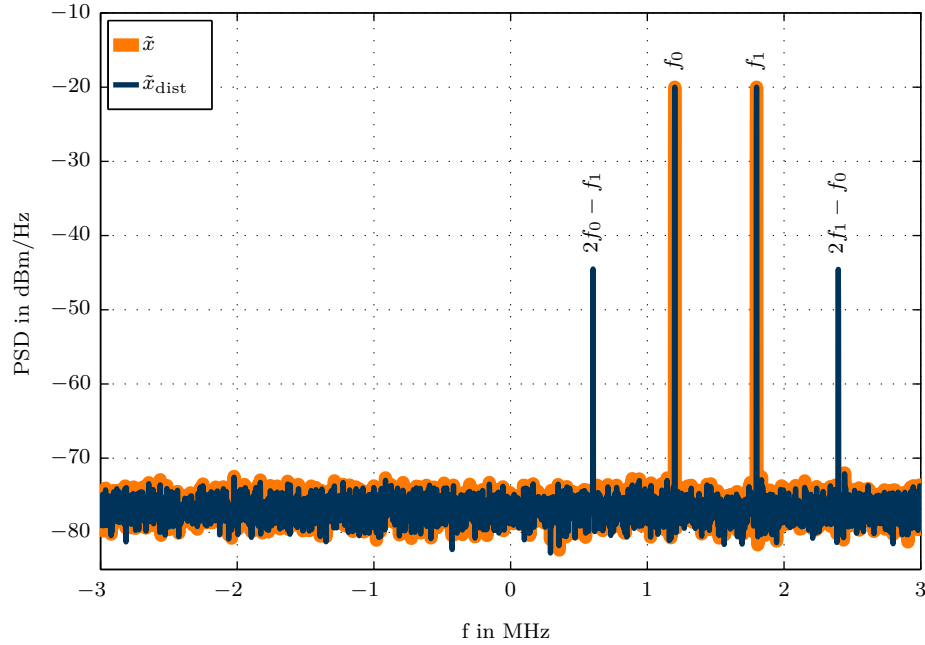


Figure 2.4: Two-tone test signal applied to the simplified memoryless, polynomial model of (2.7)

If the two-tone signal is considered as an interferer with the frequency band of about 1 to 2 MHz, the resulting IM3 products appear as out-of-band distortions and can easily hit an adjacent target band. The worst case scenario would be if the IMD products, which are induced by a *strong* interferer hit a *weak* neighbouring modulated target spectrum, what might result in an significantly deteriorated BER performance after demodulation.

## 2.3 FIR Filter

Nowadays, finite impulse response (FIR) filters have become indispensable for DSP applications as they are progressively replacing classical analogue filters [10]. The main advantages are described by their characteristics, such as linear-phase, stability and robustness to quantisation effects [18]. They offer a wide field of applications as they modify the properties of a digital signal in both time and frequency domain. The following explanations aim to give a brief overview of FIR filter methodology. More details on filter theory, design and implementation, can be found in the literature in

[10, 19, 18] and the references therein.

FIR filters as digital filters are linear time-invariant (LTI) filters, what refers to the fact that their input  $x$  is linearly convoluted with a constant impulse response  $w$  of the filter, also denoted as the filter coefficients. The impulse response of the filter is of finite length, yielding the convolution sum

$$y[n] = x[n] * w[n] = \sum_{k=0}^{M-1} w[k]x[n-k], \quad (2.12)$$

where  $*$  is the convolution operator,  $y[n]$  denotes the output and  $x[n]$  the input of the filter at a certain time instant  $n$ , also termed as snapshot  $n$ , and  $M$  is the filter order. Hence, the output of the filter  $y[n]$  is described by the sum of the actual and the past  $M - 1$  input samples weighted with the sequence of the filter coefficients  $w[k]$  with  $k = 0, \dots, M - 1$ . When applying the z-transform  $\mathcal{Z}\{w[n]\}$ , it can be obtained the z-transformed transfer function of the FIR filter

$$W(z) = \sum_{k=0}^{M-1} w[k]z^{-k}. \quad (2.13)$$

Thus, (2.12) can be rewritten, yielding

$$y[n] = \sum_{k=0}^{M-1} w[k]x[k]z^{-k}, \quad (2.14)$$

which can graphically be represented as shown in Figure 2.5 [19]. This interpretation can also be seen as a *tapped delay line*, where each delayed tap of the input samples is multiplied by the corresponding tap weight  $w[k]$  and added accordingly.

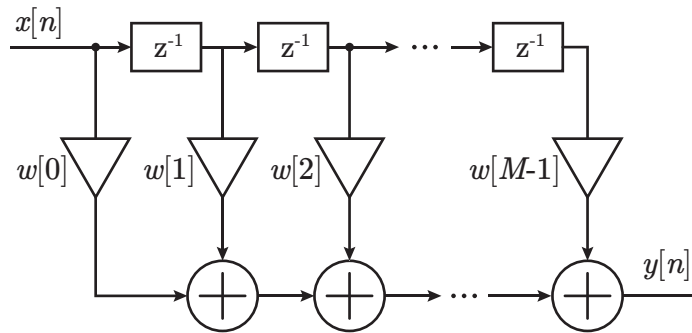


Figure 2.5: Direct form FIR filter, taken from [10]

Alternatively, (2.14) can be interpreted differently, resulting in the transposed structure of an FIR filter, as shown in Figure 2.6. Thereby, the results of the multiplication

of the actual input sample  $x[n]$  and the corresponding filter coefficient  $w[k]$  are delayed and added accordingly. This kind of FIR filter structure is preferred with regard to FIR filter implementation, since there is no shift register for holding the input samples as well as no extra pipeline stages for storing the intermediate accumulation results required [10].

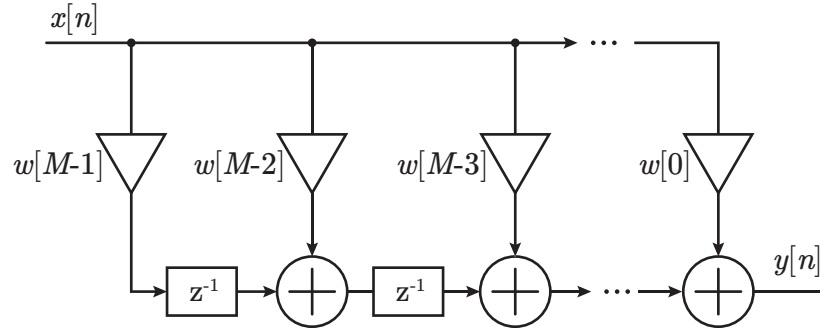


Figure 2.6: FIR filter in the transposed structure, taken from [10]

The filter design is usually performed by computer-aided engineering (CAE) tools, such as the Signal Processing Toolbox of MATLAB [20]. Hereby, the filter coefficients are created by means of the filter specifications, which are commonly defined in the frequency domain. [18] formulates the three main FIR design specifications filter order, transition width and peak ripples of the stopband- and passband. The third is determined by fixing two of the three design specifications. Besides, FIR filters can easily be designed to be linear-phase if the filter coefficient structure exhibits either a symmetric or an antisymmetric structure. The property of linear-phase is defined by the fact that all frequencies exhibit a constant group delay, which is a crucial property to many communication applications [10]. The group delay of a linear-phase FIR filter, also denoted as the settling time, is evaluated by half of the filter order  $M$  [18].

Given a linear-phase filter, the filter design degenerates into a mathematical approximation problem calling for sophisticated algorithms, which optimise the filter coefficient according to the three given design specifications. For example, the Signal Processing Toolbox of MATLAB provides several filter design methods such as equiripple, Kaiser window, or a least square FIR method [20]. The one that is used for the FIR filter design in this thesis, is the equiripple method, which shows the smallest maximum-deviation to the ideal filter frequency response compared to other linear-phase FIR filter design methods [18]. As a common equiripple design method, the Parks-McClellan algorithm, based on the Chebyshev approximation theory, is used to determine the optimal filter coefficients [21]. The filter design, by means of a bandstop

filter, is exemplary demonstrated in Chapter 4.

Designing a linear-phase filter, the filter coefficients exhibit a symmetric or antisymmetric structure, yielding two input samples to be multiplied with the same tap-weight, respectively. That is, with regard to the implementation, half of the multipliers could be saved if two input samples were accumulated before the multiplication (or subtracted in the anti-symmetric case). This will be shown more in detail in Chapter 4.

## 2.4 LMS Adaptive Filter

FIR filters show a static transfer function (2.13) as their filter coefficients are constant. However, there are many applications, where the transfer function needs to be adapted according to certain properties of the input signal, which are not known a priori or are subject to change. Thus, the direct form structure of an FIR filter is extended, as in Figure 2.5, yielding the FIR filter with adaptive filter (AF) coefficients, as depicted in Figure 2.7.

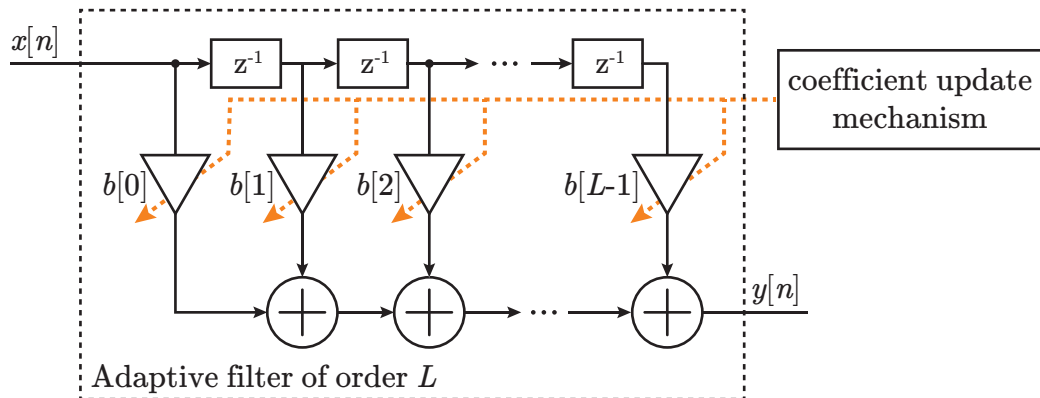


Figure 2.7: AF as an FIR filter with adaptive coefficients

The filter coefficients are thereby adapted according to a certain coefficient update mechanism. Depending on the application, [10] classifies the system configurations of an AF and divides them into four categories:

- Interference cancellation
- Prediction
- Inverse modelling
- Identification

Since the configuration used in the NONLIM algorithm is the interference cancellation, the focus of the explanations is strictly based on this type of system.

### 2.4.1 Interference Cancellation

Figure 2.8 depicts the basic configuration of the AF with respect to the application of the interference cancellation.

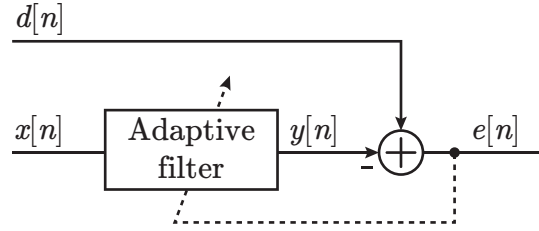


Figure 2.8: AF in the basic configuration for interference cancellation, taken from [10]

The signal  $d[n]$  represents the desired response that contains spurious interference, or distortions.  $x[n]$  denotes the input to the AF giving the filter output  $y[n]$ , which is, after convergence, adapted to the interference that is contained in  $d[n]$ . Hence, the output of the interference cancellation system is evaluated by the difference of both  $d[n]$  and  $y[n]$ , giving the error signal  $e[n] = d[n] - y[n]$ . A popular example for the application of the interference cancellation configuration is the removal of the power-line hum of 50 Hz that is contained in an information-bearing signal  $d[n]$  [10]. For example, when assuming the power-line hum to be a random white noise process, the exact probabilistic properties are not known in practice. Therefore, a simple linear filtering (e.g. by an FIR filter) is not feasible, especially if the  $d[n]$  also contains frequencies at 50 Hz. Hence, adaptive filtering is required to adjust the reference  $y[n]$  to be subtracted from  $d[n]$ , which is being achieved by the optimisation of a cost function. The most common cost function is the least mean square (LMS) function, calculated by

$$J = \mathbb{E}\{\tilde{e}[n]^2\} = \overline{(d[n] - y[n])^2} = \overline{(d[n] - \mathbf{b}^T \mathbf{x}[n])^2}, \quad (2.15)$$

where  $\mathbf{b} = [b[0], \dots, b[L-1]]^T$  and  $\mathbf{x} = [x[n], \dots, x[n-L+1]]^T$  are vectors of size  $(L \times 1)$  [10]. In order to be able to perform the adaptive process, the input signal  $x[n]$  must satisfy the two important statistical properties of being ergodic and wide-sense stationary (WSS) [22]. This allows for performing an iterative stochastic gradient optimisation approach, which is based on the Wiener-Hopf relation [10, 22]. The

objective of this adaptation is to minimise the cost function  $J$  of (2.15), what is achieved by finding the optimum values of the AF coefficients  $\mathbf{b}$ , known as the Wiener solution [22]. In order to find these, a theoretical approach is given by the Wiener-Hopf equation, yielding

$$\mathbf{b}_{\text{opt}} = \mathbf{R}_{xx}^{-1} \mathbf{r}_{dx}, \quad (2.16)$$

where the  $\mathbf{R}_{xx}^{-1}$  is the inverse of the autocorrelation matrix of size  $(L \times L)$  defined by  $\mathbf{R}_{xx} = \mathbb{E}\{\mathbf{x}[n]\mathbf{x}^T[n]\}$  and  $\mathbf{r}_{dx}$  the cross-correlation vector given by  $\mathbf{r}_{dx} = \mathbf{E}\{d[n]\mathbf{x}[n]\}$  [22]. However, there are several reasons why the Wiener-Hopf approach is not feasible in practice, which is, for example, that a high computational complexity is required to compute the correlation functions  $\mathbf{r}_{dx}$  and  $\mathbf{R}_{xx}$  and its inverse  $\mathbf{R}_{xx}^{-1}$ . Furthermore, it is not known how many samples are necessary to extract sufficient statistics [10].

### 2.4.2 Widrow-Hoff LMS Filter

The method of Widrow-Hoff is a practical approach of approximating the Wiener solution (2.16) in real-time. It neither needs to form the correlation functions nor compute the inverse of a matrix. Since it uses an instantaneous estimate of the current snapshot  $n$  of the input samples  $x[n]$  and  $d[n]$  for approximating the statistical properties, the accuracy of this approach is limited.

The LMS AF is an iterative algorithm that utilises the method of the steepest descent [22], when finding the subsequent set of filter coefficients  $\mathbf{b}[n+1]$ . Thereby, the filter coefficients of the subsequent snapshot  $n+1$  are obtained by the difference of the current coefficient set  $\mathbf{b}[n]$  and a correction, giving

$$\mathbf{b}[n+1] = \mathbf{b}[n] - \frac{\mu}{2} \nabla[n], \quad (2.17)$$

where  $\nabla[n]$  represents the true gradient pointing at the minimum of the cost function  $J$  in (2.15). The parameter  $\mu$  represents the step size controlling the convergence rate and the stability of the LMS filter [10]. However, the true gradient  $\nabla[n]$  is not known since the Widrow-Hoff approach takes only the samples of the current snapshot  $n$  into account. Thereby,  $\mathbb{E}\{\tilde{e}[n]^2\}$  is estimated by  $J = e^2$ , yielding an instantaneous estimate  $\hat{\nabla}[n]$  of the gradient  $J$  [10]. Taking the partial derivative of the instantaneous  $J$  with respect to the individual coefficient components, it can be obtained the estimate of the



gradient

$$\hat{\nabla}[n] = -2e[n]\mathbf{x}[n], \quad (2.18)$$

which can be substituted into (2.17), giving the ultimate coefficient update equation

$$\mathbf{b}[n+1] = \mathbf{b}[n] - \frac{\mu}{2}\hat{\nabla}[n] = \mathbf{b}[n] + \mu e[n]\mathbf{x}[n]. \quad (2.19)$$

Hence, the update of the coefficients is performed by adding a correction, which is composed of the cross-correlation vector  $e[n]\mathbf{x}[n]$  weighted with the step size parameter  $\mu$  to the actual coefficient set  $\mathbf{b}[n]$ .

The necessary steps of the Widrow-Hoff LMS algorithm can be summarised as follows:

1. Compute the output of the FIR filter:  $y[n] = \mathbf{b}^T[n]\mathbf{x}[n]$
2. Compute the output of the LMS filter:  $e[n] = d[n] - y[n]$
3. Update the filter coefficients for the following iteration step:  
 $\mathbf{b}[n+1] = \mathbf{b}[n] + \mu e[n]\mathbf{x}[n]$
4. Continue with step 1 for the next iteration step

At the beginning the vectors  $\mathbf{x}[n]$  and  $\mathbf{b}[n]$  are initialised to zero. At each iteration step, the new values for  $\mathbf{x}[n]$  and  $d[n]$  are shifted in, as shown by Figure 2.7 and Figure 2.8. After having computed the output of the FIR filter, the output signal of the LMS filter can be calculated. The coefficient update mechanism then evaluates the filter coefficients of the subsequent snapshot  $n+1$ . The next iteration step continues again with step 1. After a certain number of iteration steps, the coefficients converge to a particular level, indicating a converged steady-state. A metric for the accuracy of the adaptation process is the least squares (LS) (or Wiener solution) or the MSE =  $\mathbb{E}\{e[n]^2\}$  [10].

For further details on the derivations, the dependence of the step size parameter  $\mu$  and an interference cancellation example refer to the literature in [10, 22].

## 2.5 Fixed-point Arithmetic

### 2.5.1 Binary Number Representations

FPGAs internally use the binary system to represent any number and perform the algebraic operations, as all other devices that are based on digital electronic circuitry and logic gates. Binary numbers can represent both fixed-point and floating-point numbers. In general, the former benefits from higher speeds and lower costs. The latter, in turn, features a higher dynamic range and does not need any scaling [10]. As most of the DSP systems, the implementation target USRP N210 involves fixed-point arithmetic [14]. Thus, the focus is based on the fixed-point representation.

In order to be able to represent both positive and negative numbers, the sign is to be encoded beside the magnitude of the number. There are several number representations that are capable of representing signed numbers such as sign-magnitude, one's complement (1C) and two's complement (2C).

An  $N$ -bit signed integer number of the value  $X$  and the binary digits of the form  $b_{N-1}, \dots, b_1, b_0$  can be represented by the 2C. In general, a 2C is described by

$$X = -b_{N-1}2^{N-1} + \sum_{k=0}^{N-2} b_k 2^k, \quad (2.20)$$

where the bit positions  $b_{N-1}, \dots, b_1, b_0$  (either be 0 or 1) are multiplied by the corresponding powers of two and the leftmost bit, denoted as the most significant bit (MSB), with the highest weight  $2^{N-1}$  determining the sign of the value of the number. With an  $N$ -bit 2C representation, integers of the range  $-2^{N-1} \leq X \leq 2^{N-1} - 1$  can be represented. The 2C is by far the most popular number representation system being used in modern DSP systems [10], as for the USRP N210 [14]. This is due to several inherent properties such as a distinct representation of the value zero and no check being required whether to add or subtract two signed numbers. Moreover, when adding multiple signed numbers, any possible overflow of the partial sums can be ignored as long as the final result fits in the  $N$ -bit range [10].

When turning the  $N$ -bit 2C number into a 2C number with a fractional length  $F$ , the binary point is left shifted from right of the least significant bit (LSB)  $b_0$  by  $F$  positions, yielding the scaled value

$$X = -b_{N-1}2^{N-1-F} + \sum_{k=0}^{N-2} b_k 2^{k-F}, \quad (2.21)$$

where the fractional length  $F$  can be either positive or negative. Increasing  $F$  thus results in an increased precision but also reduces the maximum range that can be represented by the  $N$ -bit 2C number. The Q format specifies how many integers and fractional bits a fixed-point number has [23]. For example, Q1.16 refers to an 18-bit 2C number with 1 integer and 16 fractional bits plus the sign bit.

### 2.5.2 Bit Growth Considerations

The range of an  $N$ -bit wide fixed-point number is limited. So when performing algebraic operation, it might happen that the result does not fit in the range of an  $N$ -bit wide number any more. For example, when adding two  $N$ -bit wide 2C numbers, the word length is increased by one bit in order to avoid an overflow and a round-off error [18]. Assuming the addition of  $K$   $N$ -bit wide 2Cs, the word length of the result is increased by

$$G = \lfloor \log_2(K) \rfloor + 1, \quad (2.22)$$

whereby  $G$  is also referred to as the number of guard bits [24]. On the other hand, the multiplication of an  $N$ -bit and a  $P$ -bit wide number can potentially result in a total word length of  $N + P$  bit in order to maintain full precision [18].

If it is not accounted for a potential overflow, the 2C wraps around (either from positive to negative) or vice versa. In the following, two 4-bit 2C numbers are added to each other, obviously yielding a result that does not fit in the 4-bit range

$$8_{10} + 4_{10} = 0111_{2C} + 0100_{2C} = 1011_{2C} = -5_{10}$$

Hence, if an addition of either two positive values or two negative values gives a negative or positive result, respectively, an overflow has occurred. In DSP applications, overflows must be strictly avoided, since these represent abrupt jumps in the waveform and result in catastrophic distortions.

However, maintaining the word length within the DSP processing chain at a constant width is essential. For this reason, scaling must take place for fitting the result in the range, which is commonly being established by an arithmetic bit-shift. For each bit positions  $r$ , which the binary sequence is arithmetically right shifted, the value of the number is scaled by  $2^{-k}$ . With respect to a 2C number, an arithmetic right-shift means that the vacated leftmost bit positions are filled with the corresponding sign bit, ensuring also a proper scaling of negative numbers.

### 2.5.3 Rounding Methods

Performing a bit-shift of  $r$  bit positions always means that the  $r$  LSBs are truncated. That is, the number to scale is rounded off independently of the truncated values. The advantage of the bit truncation is that almost no additional hardware resource are used. However, the drawback is that this inevitably introduces a bias towards the negative by 0.5 [24].

[24] summarises different kinds of rounding methods such as non-symmetric rounding to positive/negative, symmetric rounding to highest magnitude/zero, approximation of symmetric rounding and convergent rounding. The method that is mainly being used within the scope of this implementation is the non-symmetric rounding to positive, adding a binary 1 to the rightmost  $r$ -th bit position  $b_{r-1}$ . Although a bias of  $2^{-(r+1)}$  is introduced, this type of rounding constitutes a viable rounding method since almost no hardware resources are required [24].

## 2.6 Field Programmable Gate Array

A field programmable gate array (FPGA) is an integrated circuit (IC) that is aimed to be reconfigured by the end-user. FPGAs have been evolved from the application-specific integrated circuit (ASIC) technology, which is, in turn, fabricated by manufacturers for a particular purpose. That is, the functionality of the designed ASIC chip is fixed after the fabrication process and cannot be changed any more. ASICs are very efficient in terms of performance and energy consumption but are very expensive and complicated if fabricated in small quantities. In contrast, FPGAs provide the digital circuit designer, a highly flexible, fast and cost-efficient option by means of implementing a digital architecture. Furthermore, [25] states the following (main) benefits of FPGAs:

**Software defined design** The behaviour of the digital circuit is modelled using a HDL, usually VHDL or Verilog. The so-called electronic design automation (EDA) tool translates the HDL model into a digital circuit and maps it onto the FPGA architecture at hand.

**Parallelism** Algorithms can run in a parallel manner on FPGAs, similar to multi-path analogue circuits. This represents a tremendous benefit, compared to the consecutive task computation of processor-based systems.

**High speed** FPGAs are usually operating with high clock rates of hundreds of megahertz. Thereby, digital designs run at very high speeds.

Hence, using FPGAs in the field of DSP in communication systems suits particularly well, since typical mathematical operations like the fast Fourier transform (FFT), interpolation, decimation and filtering call for high computational efforts. Additionally, the requirement of high-speed processing is given due to the demand of real-time applications, such as SDR, whereby a pure processor-based DSP would most likely be overloaded.

### 2.6.1 Architecture

Contemporary FPGAs are highly dense single chip devices that contain multiple millions of gates. In recent years, the field of FPGA has evolved significantly, whereby architectural components are being steadily improved [25]. The trend continuous towards IC design, combining all components that are usually in a general purpose computer. To elaborate all of the existing FPGA architecture would go far beyond the scope of this thesis. Further information on a survey of various IC architectures can be found in the literature [25, 26, 10, 27]. The focus is instead based on the Xilinx Spartan-3A DSP FPGA, which is the implementation target of the mitigation algorithm to be established and can also be considered as an up-to-date FPGA device. Its fundamental elements are the configurable logic blocks (CLBs), digital clock manager (DCM) blocks, block RAM, input/output blocks (IOBs) and DSP48A slices. Each element has an associated switch matrix, interconnecting it to other elements and thereby forming a rich routing network [28]. The arrangement of the elements is depicted in Figure 2.9.

CLBs are arranged as a dense matrix array and form the main logic resource of an FPGA. Those are also referred to as *fabric*. One CLB consists of four slices in total, whereby one slice encompasses two 4-input lookup tables (LUTs), two flip-flops (FFs) as its storage elements, two wide-function multiplexers, as well as carry and arithmetic logic [29]. The LUTs of two of the four slices of a CLB can be used as distributed random-access memory (RAM) or as shift register logic (SRL), both yielding a 16-bit memory element, what is dictated by the 4-input LUT structure at hand. Logical functions are implemented using a LUT, forming a truth-table. For example, a 4-input LUT therefore yields a memory with  $2^4 = 16$  entries addressed by the inputs. The carry and arithmetic logic, in combination with the dedicated wide-functions multiplexer, interconnect a LUT with other LUTs and CLBs, which enables the implementation

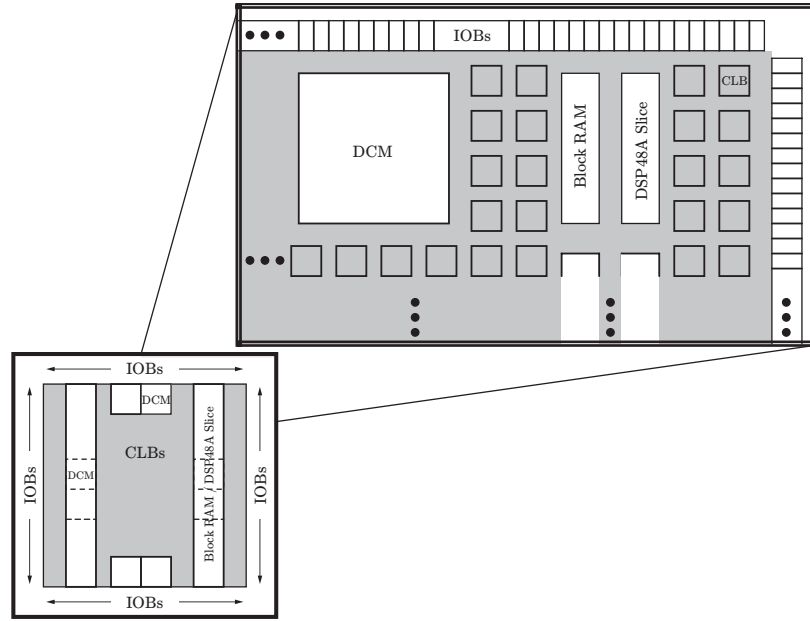


Figure 2.9: Exemplary FPGA architecture by means of the Xilinx Spartan-3A DSP, taken from [28]

of more complex arithmetic [29]. This type of memoryless logic is considered to be combinatorial and the outputs strictly depend on the present value of the inputs. Additionally, FFs, also denoted as registers, serve the purpose of synchronising data to clock signals in order to form sequential logic [29].

Pursuing a synchronous design, a simultaneous switching of all synchronous elements (e.g. FFs) would be absolutely necessary. The global clock is distributed over a dedicated low-capacitance interconnect network, in which its input takes place at the edges of the FPGA. The effect of varying signal propagation delays of the clock signal, inferred from different signal paths, is called *clock skew*. It is pursued a widely equal clock distribution [26]. In practice, however, perfectly synchronous switching can never be achieved. DCMs that are positioned at the edges of the FPGA, which compensate for the *clock skew* and are therefore essential elements of the global clocking infrastructure [29]. Further functions of the DCMs are phase-shifting a clock signal, multiplying or dividing an incoming clock frequency, or conditioning and converting a clock signal [29].

IOBs form a ring of physical connector pins around the FPGA device and represent a highly configurable, high-performance physical interface to other semiconductor devices (e.g. memory, processors, ADCs, DACs) that are also part of the PCB-based design. Depending on the application, different input/output (I/O) standards, such as low-voltage differential signalling (LVDS), peripheral component interface (PCI), low

voltage complementary metal oxide semiconductor (LVCMOS) and gunning transceiver logic (GTL), can be selected for performing data transmission or signalling with outer elements [25]. They require to be run with different specifications such as current, voltage, I/O buffering, and termination techniques, either be configured as a bidirectional or unidirectional link [29].

When storing large amounts of data, the usage of CLBs as distributed RAM can be considered as waste of resources. Instead, dedicated RAM blocks within the FPGA architecture, which are thus called block RAM, are more appropriate and act as large storage elements that are well integrated into the FPGA design. The 18 kbit-large, dual-port RAM blocks of Xilinx Spartan-3 architectures offer a wide variety of configurations that can be selected including RAM, read-only memory (ROM), first in, first outs (FIFOs), large LUTs or SRLs [29]. With respect to Spartan-3 architectures they are positioned together with DSP48A blocks in a common column in order to keep the routing delays low when using both elements jointly. DSP48A blocks are composed of an embedded 18 bit  $\times$  18 bit, signed hardware multiplier and a pre- and postadder that are optimised for DSP applications. They are discussed in more detail in Chapter 4.

## 2.6.2 Design Flow

In order to implement a digital circuitry onto an FPGA, a certain process should be followed. Vendors of the FPGA devices are providing software programs, denoted as EDA tools, facilitating the implementation process. Xilinx' EDA tool is called *ISE Design Suite*, on which the focus will be pointed at in the following. The design flow of the implementation process is divided into the sub-processes design entry/synthesis, simulation and design implementation, as illustrated in Figure 2.10. Individual sub-processes can be executed on their own but are generally combined by the EDA tool in order to alleviate the overall design flow. Figure 2.10 depicts, for sake of simplicity, a summarised design flow of the *Xilinx ISE Design Suite*. A full documentation of the FPGA design and implementation methodologies can be found in the corresponding Xilinx documentations [30, 31, 32, 33].

### Design Entry and Synthesis

The design entry constitutes the initial starting point, how the implementation of digital circuits can be established. This can be done by either pursuing a large-flat or a hierarchical design approach. The former combines every single module in one project file, whereas the latter describes a top-level module that instantiates lower-level

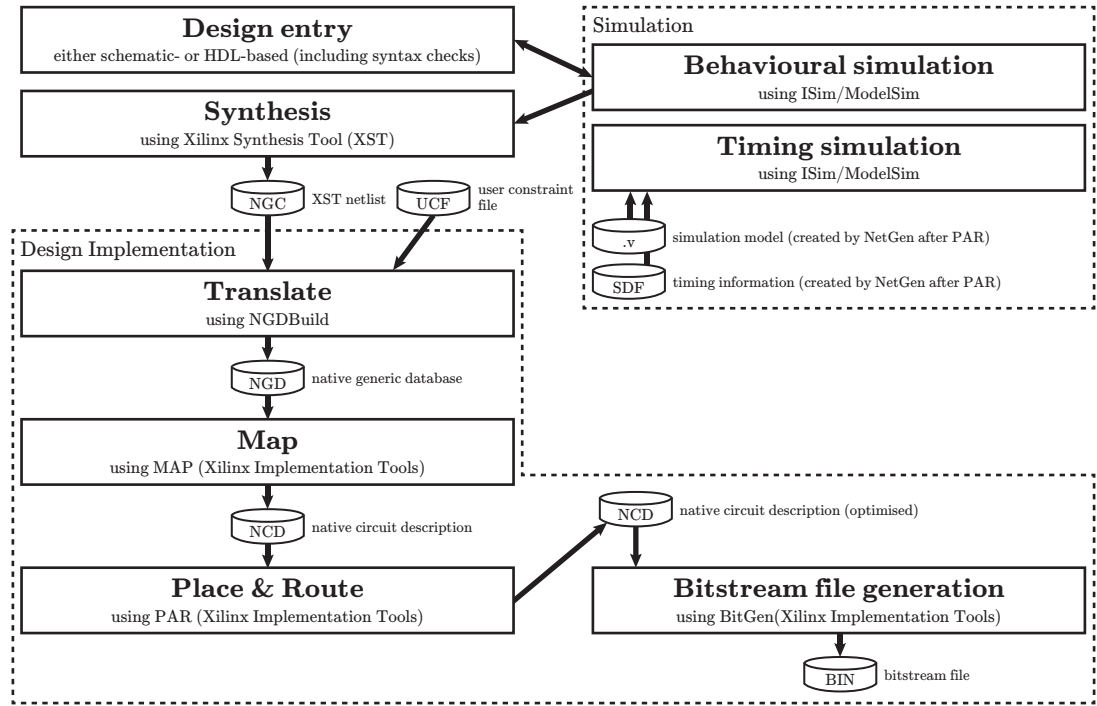


Figure 2.10: FPGA design flow

sub-modules. In general, the hierarchical design is preferable for an easier and faster verification and simulation. Furthermore, it enables several engineers to work together on a project. It offers an increased design compilation speed, creates a design that is easier understandable and allows for a more efficient design flow [33].

Two main methodologies are provided in the *Xilinx ISE Design Suite*: A graphical schematic-based and a textual HDL-based design. The difference between both is defined by the interconnection of the sub-modules. Using a schematic-based design, the wiring is set up using graphical elements via drag-and-drop, where the HDL-based solely uses textual wiring. Hence, the latter allows for a fully described data flow being processed and transferred between the registers in a digital circuit. This type of abstraction level is thus called register transfer level (RTL) [34]. The sub-modules that are instantiated within the top-level design can either be HDL-based modules or Intellectual Property (IP) cores, being generated by the *CORE Generator* tool. An IP core constitutes a pre-defined module with a specific function, which can easily be instantiated and reused within a design. It is either delivered as HDL-code or as a synthesised netlist, whereby the latter offers the advantage of making reverse engineering difficult in order to protect the intellectual property of the author [26]. Using IP cores speeds up the development process significantly [26]. The *CORE Generator* is a graphical design tool that enables the designer to create customised high-level IP mod-



ules, such as memory and storage elements, communication interfaces, DSP-processing blocks etc. [35]. The most common HDL languages are VHDL (IEEE 1076-1993) and Verilog (IEEE 1364-2001), being supported by most of the EDA tools. The choice between the two is dependant of the designer. VHDL is more verbose and implies more language constructs, such as strong typing. Verilog is, in turn, more simple and its syntax is similar to the one of the C programming language [36]. Since the existing code of the USRP FPGA firmware is written in Verilog, the implementation is carried out using Verilog [37].

Once the design is created, it can be simulated on the RTL with an HDL simulator, e.g. *ISim*, which is also integrated within *ISE*. At this point, only the functional behaviour of the written HDL code is simulated without taking any timing information of the interconnections into account. Instead, only a static timing analysis is performed where fixed delays are assumed. This type of simulation is called a functional or behavioural simulation.

As a next step, the Xilinx Synthesis Tool (XST) performs the synthesis of the HDL code that is composed of the sub-processes HDL parsing, HDL synthesis and low-level optimisations. When conducting the parsing process, the HDL code is first checked for syntax errors. During synthesis, XST analyses the code in order to recognise design building blocks, extract finite state machines (FSMs) and to share resources. This is done according to certain design constraints, such as the optimisation goal and effort that are also input to the synthesis process [32]. A low-level optimisation is followed, where inferred building blocks and general logic is transformed into a technology-specific NGC netlist, which consists of specific components, such as LUTs, carry logic, I/O buffering, multiplexers, RAM, DSP48A slices [32]. To investigate the synthesis results, the schematic viewer is able to show schematics of both the pre-synthesis RTL view and the post-synthesis, technology-specific view. Furthermore, a synthesis report summarises the amount of resources used by the implemented design in comparison to the total available amount of the target architecture.

## Design Implementation

The design implementation is composed of the sub-processes translate, map, place and route and the bitstream file generation. Therefore, the previously generated netlist is mapped onto the device architecture. At first, the translate process is established using the tool *NGDBUILD*, which reads the input NGC netlist and constraint files such as the user constraints file (UCF) and creates a native generic database (NGD) file containing the hierarchical logic components and the lower-level NGD primitives

[31]. The UCF is an ASCII file that allows for incorporating user constraints, such as timing and I/O, and placement constraints [30]. Graphic tools such as the *Constraints Editor* or *PlanAhead* (which are integrated in *ISE*) help to set up these constraints.

From then on, the *MAP* program performs the mapping of the logical design to the primitive components (e.g. CLBs or IOBs) of the FPGA target architecture. The output is a native circuit description (NCD) file, which comprises the physical representation of the design being fit onto the resources of the FPGA, and a map report (MRP) listing the utilised hardware resources of each individual sub-module [31].

Then, the NCD file is placed and routed using *PAR* that again gives an output of an optimised NCD file. The timing-driven *PAR* is based on the timing constraints specified by the UCF. The *PAR* process interacts with an integrated Xilinx timing analyse software in order to optimise the placing and routing until the imposed timing constraints are met [31]. Upon having placed and routed the design, a netlist containing the simulation model and a standard delay format (SDF) with the timing information can be generated using the tool *NetGen*. Hence, a back annotated timing simulation can be conducted implying the routing and component delays induced by the implementation process. Introducing delays could lead to unexpected results. Thus, a timing simulation could reveal timing violations as well as constitutes an important step of the implementation process [33].

Finally, the tools *BitGen* or *iMPACT* take the fully routed NCD (as well as other target device related information) to output a bitstream (BIT) or a binary (BIN) file (containing no header information). The process of programming the binary files onto the FPGA is called configuration, which is loaded into an internal or external non-volatile memory. From there, the configuration bitstream is loaded at each power start-up [38]. With respect to the USRP N210, the FPGA firmware is stored into an on-board flash RAM and can be configured using the ethernet interface of the device [14].

### 3 The NONLIM Algorithm

The main idea of the non-linearly induced interference mitigation (NONLIM) algorithm is to mitigate the IMD products in the digital back-end, which were originally generated in the analogue RF front-end [3]. With respect to the DCR, as illustrated in Figure 2.1, the compensation of non-linear distortions takes place in the digital back-end where the RF signal has already been down-converted to BB. When dealing with a receiver terminal, the original transmitted signal is not known to the receiver. In this case, the non-linear distortions are artificially regenerated based on the simplified memoryless, polynomial model of (2.9), which serves as a reference non-linearity. As a drawback of this technique, only non-linear distortions that originate from strong block signals within the down-converted BB can be reproduced. The obtained reference distortions are subsequently subtracted from the distorted signal using an adaptive filter (AF). For illustrative purposes, the use case scenario is composed of a strong two-tone block interferer and a weak, Gaussian-like shaped target spectrum, as depicted in Figure 3.1.

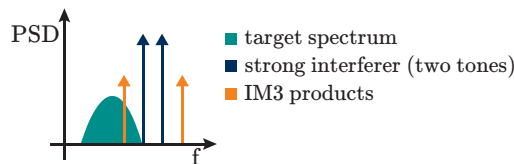


Figure 3.1: Illustration of the use case scenario with a strong two-tone blocking interferer and a weak, Gaussian-like shaped target spectrum

The target signal essentially represents any modulated signal (e.g. modulated with Gaussian minimum-shift keying (GMSK) or binary phase shift keying (BPSK)). Accordingly, one of the resulting third-order intermodulation (IM3) products hits the target frequency band.

Figure 3.2 shows a block diagram, which explains the principle of the NONLIM algorithm by means of the aforementioned two-tone scenario. Basically, the algorithm performs the mitigation operating in two separated branches, the upper and lower branch of Figure 3.2, which are, in the following, referred to as the *desired* and

the *reference* branch, respectively. The algorithm is based on time-domain filtering only and consists of several FIR bandpass/-stop filters, a reference non-linearity, and a least mean square (LMS) AF, which adjusts the reference distortions to the actual distortions in the desired branch. The instances are denoted by the names *bsd\_filter*, *bpr\_filter*, *refmodel* and *lms\_filter*, where the acronyms bandstop-desired (BSD), bandpass-reference (BPR), and bandstop-reference (BSR) correspond to their types and appearances in the desired and reference branch, respectively. The I/Q components are processed separately (apart from the complex *bpr\_filter* and the *refmodel*) in order to facilitate the implementation procedure. In the following, any signal provided with the tilde-operator, such as  $\tilde{x}[n]$ , denotes the complex-valued signal given by its corresponding I/Q components, namely  $\tilde{x}[n] = x_I[n] + jx_Q[n]$ .

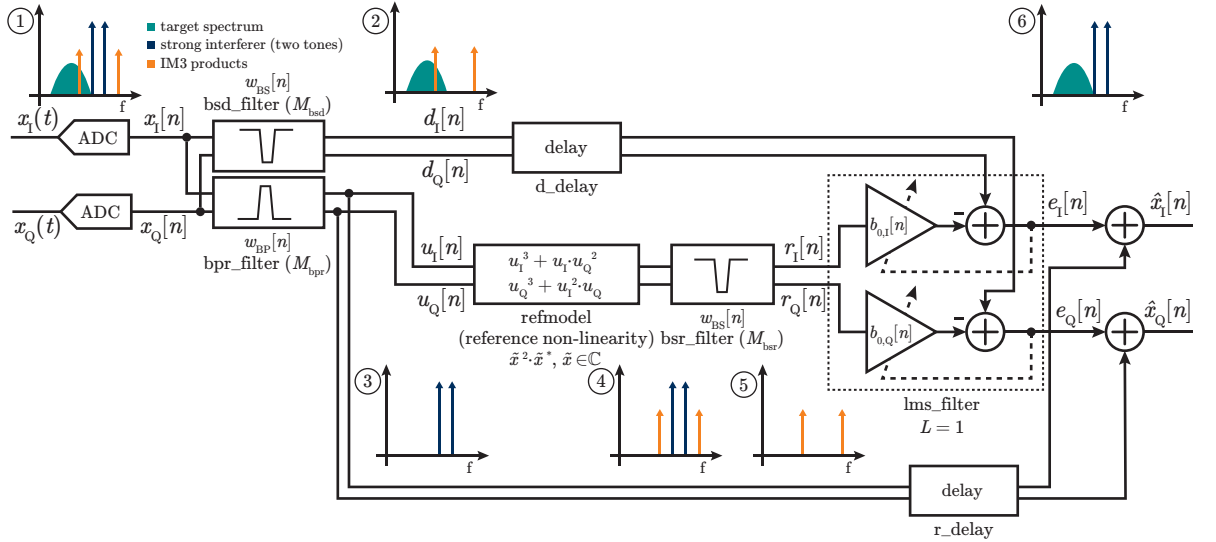


Figure 3.2: Block diagram of the NONLIM algorithm by means of the two-tone example

### 3.1 Reference Branch

The function of the reference branch is to provide reference distortions to the AF. First, a band-split filtering stage isolates the interferer from the wideband spectrum, yielding signals  $\tilde{d}[n]$  and  $\tilde{u}[n]$ , according to (2) and (3) in Figure 3.2, respectively. The *bpr\_filter* is characterised by the interferer BW, the unit passband gain and a given stopband attenuation, forming the complex coefficients  $w_{BP}$  of finite filter order  $M_{bpr}$ . The *bpr\_filter* needs to be complex in order to avoid mirror frequency components,

which are caused by I/Q-imbalance, entering the reference branch. The choice of the stopband attenuation depends both on the receiver sensitivity and the strength of the interferer. The resulting signal components  $u_I[n]$  and  $u_Q[n]$  are then being fed to the reference model of (2.9), where the reference non-linearity is being generated. The respective mathematical operation for the I- and Q-components are obtained from multiplying out the latter term  $3a_3\tilde{x}^2\tilde{x}^*$  of (2.9), yielding the so-called reference non-linearity

$$\begin{aligned}\tilde{u}_{\text{rnl}} &= 3a_3\tilde{u}^2\tilde{u}^* = 3a_3(u_I + ju_Q)^2(u_I - ju_Q) = 3a_3(u_I^2 + 2ju_Iu_Q - u_Q^2)(u_I - ju_Q) \\ &= \dots = 3a_3[(u_I^3 + u_Iu_Q^2) + j(u_Q^3 + u_I^2u_Q)],\end{aligned}\quad (3.1)$$

where  $a_3$  is the real-valued polynomial coefficient (the time variable  $n$  was skipped for convenience). Since the exact coefficient  $a_3$  is not known a priori, the factor  $3a_3$  can be completely skipped without affecting the performance of the mitigation. The output of the reference model block, as shown in (4) in Figure 3.2, still contains the actual interferer. Hence, a *bsr\_filter* (of finite filter order  $M_{\text{bsr}}$ ) with an inverse passband characteristic to the preliminary *bpr\_filter* is used to obtain the IM3 products only. Since the frequency components of the interferer are only present on the positive frequency axis, a resource-efficient real bandpass realisation is sufficient. (5) in Figure 3.2 now serves as the regenerated reference non-linearity  $\tilde{r}[n]$  for being subtracted from the distorted signal in the desired branch.

## 3.2 Desired Branch

The desired branch represents the actual signal path from where the regenerated reference non-linearity is to be subtracted. Before doing that, the spectrum band of the interferer first has to be filtered out by applying a *bsd\_filter* (of finite order  $M_{\text{bsd}}$ ). The passband of the filter complies with the frequency band of the interferer and its characteristic is inverse to that of the *bpr\_filter*. The order of magnitude strongly depends upon the strength of the interferer and its produced IM3 products. Details about the actual filter design are investigated in Chapter 4. The resulting filtered signal  $\tilde{d}[n]$  now consists of the desired signal and the IM3 distortions, according to (2) in Figure 3.2. At this point, it must be considered that both the desired and the reference branch are accounting for a different processing delay, which can be divided into two kinds. First, there is a pure processing delay that an individual sample requires to travel through the processing chain. Second, another delay is added by the FIR filters until a valid

output is provided, resulting in the settling time delay of the FIR filters. Due to a time offset between both the desired and the reference branch, a proper subtraction of the reference distortions  $\tilde{r}[n]$  from the original distorted signal  $\tilde{d}[n]$  by the LMS AF is unlikely. Thus, the delay block  $d\_delay$  is employed for perfectly aligning both branches. The number of samples of the total delay  $d\_delay$  equates to

$$\begin{aligned} n_{d\_delay} &= n_{\text{ref,delay}} - n_{\text{des,delay}} \\ &= (n_{\text{r,proc}} + n_{\text{ref,settle}}) - (n_{\text{des,proc}} + n_{\text{des,settle}}), \end{aligned} \quad (3.2)$$

where  $n_{\text{ref,delay}}$  and  $n_{\text{des,delay}}$  correspond to the delays of the reference and the desired branches, respectively. How the delay is eventually realised, strictly depends on the implementation, what is looked at more detail in Chapter 4.

### 3.3 LMS Adaptive Filter

The polynomial coefficient  $a_3$  of the reference non-linearity is not known a priori and thus the reference distortions will most likely not match those contained in  $\tilde{d}[n]$ . Hence, the task of the LMS AF [22, 10] is to adjust the reference distortions in amplitude and phase and to subtract them from  $\tilde{d}[n]$ . The LMS filter is composed of an FIR filter of finite order  $L$ , whose coefficients are iteratively adapted by the filter coefficient update section, following the Widrow-Hoff algorithm as demonstrated in Chapter 2. The output of the LMS filter  $e[n]$  converges to the minimum mean square error (MMSE), known as the Wiener solution [22].

Once the AF has converged, the reference distortions  $\tilde{r}[n]$  are ideally adapted to the actual non-linear distortions in  $\tilde{d}[n]$ . The choice of the filter order  $L$  and the step size  $\mu$  largely depends on the signal at hand, which will also be addressed in Chapter 4.

The bandsplit filtering stage employed by the *bsd\_filter* and *bpr\_filter* isolates the interferer band from the rest of the spectrum. Thus, the output of the LMS filter does not contain the interferer band any more. This is not desired if the BB should merely be cleansed of non-linear distortions. Therefore, the filtered out interferer signal  $\tilde{u}[n]$  is added back to  $\tilde{e}[n]$  in order to retrieve the former spectrum of  $\tilde{x}$  that is now cleaned of the IM3 products, as depicted by ⑥ in Figure 3.2. Due to the settling time delay introduced by the reference branch, another delay block  $r\_delay$  must be integrated in order to add the samples at the appropriate instant in time. The amount of samples, by which the addition of the interferer signal  $\tilde{u}[n]$  must be delayed, accounts for the sum of the reference branch' settling time delays after the bandsplit filtering stage,

yielding

$$n_{\text{r\_delay}} = n_{\text{des\_settle}} = n_{\text{bsr\_settle}}. \quad (3.3)$$

### 3.4 Simplifications and Limitations

Applying this form of the NONLIM algorithm also implies some simplifications and limitations.

The NONLIM algorithm does not take clipping distortions, which are caused if the ADCs becomes overloaded, into account. Thus, the interferer signal is chosen to show a low peak-to-average power ratio (PAPR) in order to strictly avoid clipping distortions.

Since AM/AM distortions of the LNA are assumed to dominate, correlations between the I/Q can be neglected and a separated processing of both I/Q components is sufficient. This has been demonstrated in [3], where separated I/Q processing leads to considerable improvements in the mitigation performance, performing pure offline MATLAB processing. Consequently, this implementation involves separated I/Q processing in order to facilitate the procedure of implementation.

Nevertheless, a complex implementation might have advantages than separated I/Q processing. Practically, a correlation between the I/Q components might be introduced due to I/Q imbalance, which could affect the mitigation performance of the LMS AF. Therefore, a complex-valued implementation of the LMS filter might outperform a real-valued LMS filter implementation. For achieving the same mitigation performance, a smaller LMS filter order  $L$  could be used and therefore precious hardware resources could be saved. To the end of Chapter 4, a potential complex-valued LMS implementation is discussed and both variants are compared to each other.

As a further simplification, any cross-modulation between the target signal and the interferer is ignored [4]. According to [17], the strength of the cross-modulation products depends on the relative strengths of the underlying signals. With respect to the weak desired signal, any resulting, additional intermodulation products are expected to be negligibly small so that they could be ignored. Hence, the NONLIM algorithm focuses on the mitigation of the dominant IM3 products.

## 4 Implementation

### 4.1 Target Architecture

The NONLIM algorithm is to be implemented on the SDR-platform USRP N210 [12]. It is equipped with the FPGA Xilinx Spartan-3A DSP (XC3SD3400A), which is perfectly suited for DSP-applications due to additional block RAM and DSP48A slices [28].

#### 4.1.1 DSP48A Slice

DSP48A slices are provided as many DSP applications involve a multiplication followed by an addition [39]. Therefore, they basically consist of a pre-adder, a signed 18x18-bit multiplier, as well as a post-adder without the use of general FPGA fabric, as illustrated in its simplified form in Figure 4.1.

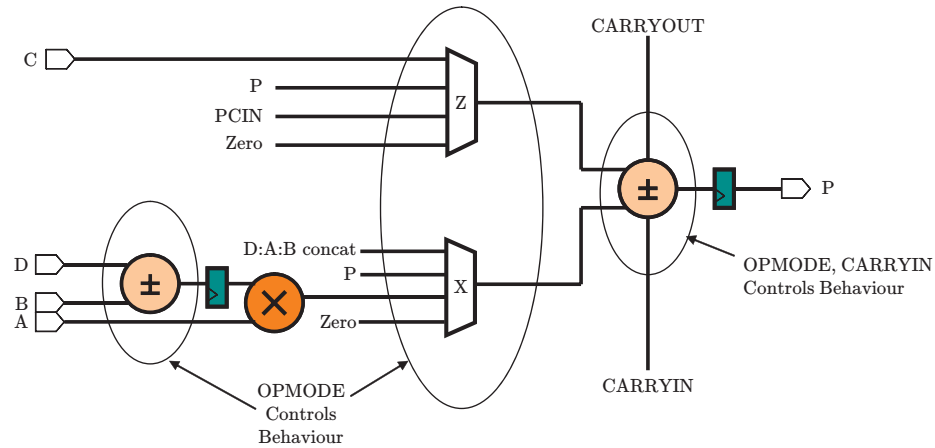


Figure 4.1: Simplified DSP48A slice model, taken from [39]

A DSP48A slice has two 18-bit input ports A, B and an input port C of the width 18 bit. Depending on the math operation to be performed, the 8-bit OPMODE input controls which input of the multiplexers X and Z is selected and whether the pre- and post-adder should add or subtract. The dedicated resources PCIN/PCOUT, as well as



B and BOUT (not shown in the figure), are interconnected to the adjacent DSP48A slices, which enables an efficient routing when cascading multiple DSP48A slices as often required by FIR filter implementations. Another pipeline register with an independent clock enable (CE) is employed between the data ports and the other elements, allowing a time-multiplexed usage of the DSP48A slices. The use of the DSP48A slices allows for an efficient implementation of applications like FIR filters, complex multiplications, multi-precision multiplications, complex Multiply-Accumulates (MACs) and adder cascades [39]. The configuration of the cascaded inputs and the pipeline registers allows to trade-off between high performance (increased throughput, multiple parallel slices) versus data latency (time-multiplexed slice usage). Further information on DSP48A slices is available in [39].

### 4.1.2 Available Resources

The implementation of custom FPGA logic into the USRP N210 is suited particularly well because a large amount of resources is available. The amount of resources utilised by the existing FPGA design and of the total resources being available is summarised in Table 4.1.

Table 4.1: Resource utilisation of NONLIM

Resource type	Amount used	Total Amount	Available
Flip-Flops (FFs)	20 007	47 744	59 %
4-input LUTs	30 500	47 744	37 %
Block RAM	41	126	68 %
DSP48A	31	126	76 %

Accordingly, 95 out of 131 DSP48A slices are available. The amount of free FFs and LUTs accounts for 59 % and 37 % of the total amount, respectively, which offers sufficient room for implementing custom FPGA logic.

### 4.1.3 Analogue-to-digital Converter

In order to convert the analogue signal into the digital domain, the USRP N210 is equipped with a dual channel ADC ADS62P45 from Texas Instruments [40, 14]. It provides a sample rate of 100 MSPS and a resolution of 14-bit, giving a 2C binary output. The spurious-free dynamic range (SFDR) of the ADC amounts for 88 dBc. Further details of the ADC can be found in [40].

#### 4.1.4 RF Front-end

Different types of RF front-ends can be used in conjunction with the USRP N210. Due to the modularity of the USRP platform, the daughterboard, on which the RF front-end resides, is plugged onto a socket of the USRP motherboard. The RF front-end, which is used within the scope of this work, is the WBX front-end, provided by Ettus Research [14].

The behaviour of second- and third-order distortions of a non-linear system can be characterised by the so-called second-order intercept point (IP2) and third-order intercept point (IP3). Both are fictitious points commonly determined by performing two-tone measurements with the non-linear device at hand. They are defined by the point, where the output power of the fundamental, linear gain of the amplifier equals the power of the second- and third-order distortions [17]. The input power given at the IP2 and IP3 is referred to as the input-referred second-order intercept point (IIP2) and input-referred third-order intercept point (IIP3). [3] has carried out practical measurements to determine the IIP2 and IIP3 of the WBX front-end and measured them to be 58.9 dBm and 13.9 dBm, respectively, being almost frequency-independent [3]. Therefore, the IM3 distortions are dominating, which is in accordance to the assumptions in which the simplified polynomial model of (2.7) is based on.

#### 4.1.5 Analysis of the Existing DDC Chain

In order to implement the NONLIM algorithm, the existing FPGA firmware has to be analysed first. In Chapter 1, it has been demonstrated how a received signal is down-converted to BB. The ADCs digitise the BB samples at a sampling rate of 100 MHz with a resolution of 16 bit per channel, exceeding the maximum transfer rate of GigE. Therefore, the sample rate must be decimated by at least a factor of 4 in order to be able to transfer the data at a maximum rate of 25 MHz. A block diagram of the USRP FPGA firmware is depicted at the top of Figure 4.2. Basically, it is composed of a COordinate Rotation DIgital Computer (CORDIC), a cascaded integrator-comb (CIC) decimator and a two-stage halfband decimator [41, 42, 43]. These constitute computationally efficient realisations that are commonly used for hardware implementations [10].

The CORDIC can be understood as a numerically controlled oscillator (NCO) that is controlled by the Universal Hardware Driver (UHD) interface of the host PC. Most of the RF front-ends do experience a finite frequency accuracy when tuning into the desired centre frequency. For example, the WBX front-end incorporates a frequency

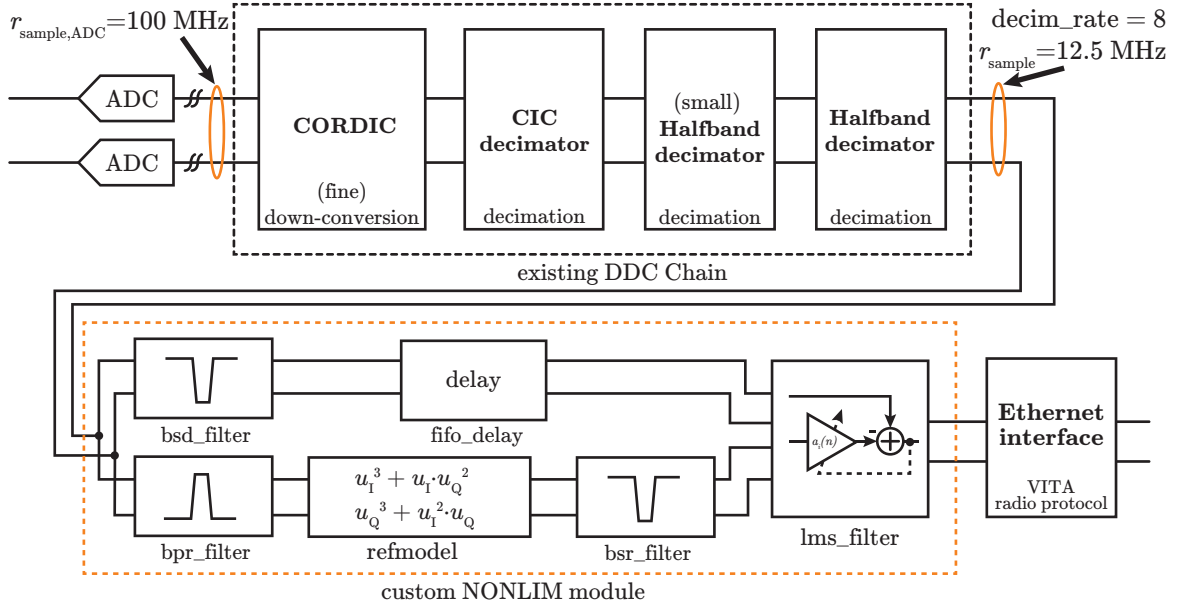


Figure 4.2: Existing DDC chain with appended NONLIM algorithm

accuracy of 2.5 ppm [14]. In order to account for that, the host PC sends a command containing a phase correction over the UHD interface to fine-tune the desired centre frequency. It also offers the options of instantaneous frequency-hopping (the RF front-end implies a certain settling time), DC offset correction and multi-channel use of a single RF front-end [44]. CORDIC is an iterative algorithm performing approximating computations of the trigonometric functions. The employed 20-stage CORDIC block iteratively adjusts the phase by adding or subtracting constant bit-shifted phase corrections, whereby the accuracy increases at each iteration step [45].

The CIC filter, in combination with two cascaded halfband filters (HBFs), performs the decimation of the input sample rate. The CIC filter is commonly used for hardware-efficient implementations as an interpolation and decimation filter and uses solely additions and subtractions as its arithmetic operations [46]. It benefits from the freely programmable decimation rate and its hardware utilisation efficiency. However, it yields firstly a drooping passband response, where the aliased spectral components appear in the passband after decimation, and secondly a significant passband roll-off. To account for that, two HBFs are cascaded after the CIC filter [46].

Halfband filters are symmetric FIR filters with every second coefficient being zero. Compared to CIC filters, the passband response of HBFs has a linear characteristic and aliasing spectral components, appearing in passband, are reduced [43]. As a drawback, hardware multipliers are required to perform the necessary multiplications that are part of the FIR structure. However, compared to usual symmetric FIR filters,

only half of the amount of multipliers are needed due to the fact that every second coefficient equals zero. In the USRP DDC chain, cascading the HBFs after the CIC filter serves the purpose of compensating the passband roll-off. The order of the two HBFs is chosen to be 2 and 8, what corresponds to the number of non-zero coefficients. Therefore, the second HBF implies a more flattened passband response. The reason for dividing the halfband section into two stages is that the second HBF constitutes a more area-efficient realisation, whereby the hardware multipliers are being used in a time-multiplexed manner. [44]

It is highly sophisticated that both cascaded HBFs are enabled so that the passband response gets linearised at its best. This implies the decimation rate being chosen as a multiple of 4. A logic of the UHD interface controls which of the HBFs is enabled, depending on the factor *decim\_rate* [44]:

- multiples of 4: first and second HBF enabled,  
 $\text{CIC\_decim\_rate} = \text{decim\_rate}/4$
- multiples of 2: first HBF enabled, second HBF deactivated,  
 $\text{CIC\_decim\_rate} = \text{decim\_rate}/2$
- odd factors: both HBFs deactivated, high CIC roll-off,  
 $\text{CIC\_decim\_rate} = \text{decim\_rate}$

In the case of an odd decimation rate, both HBFs are deactivated resulting in a high CIC roll-off, what should be strictly avoided. If, for any reason, an odd sampling rate is required, further decimation is therefore recommended to be performed on the host PC.

With respect to the implementation of NONLIM, a fixed decimation rate of 8 is chosen, corresponding to a decimated sample rate  $r_{\text{sample}} = 12.5 \text{ MHz}$ . Accordingly, the requirement of accepting and processing samples at a certain rate is relaxed with regard to the sub-modules design of NONLIM. Therefore, the implementation of the NONLIM module is facilitated by placing it after the existing DDC chain, as illustrated at the bottom of Figure 4.2.

## 4.2 Implementation Design Flow

When modelling digital circuits in HDL, designers face several challenges. First, the integration of high-level DSP algorithms into FPGAs must be understood. This includes mapping basic arithmetic operations onto on-chip elements, such as embedded

hardware multipliers, adders/subtractors, FFs, LUTs or other primitives [47]. Thereby, a limited number range has to be considered, as a fixed-point representation system is most commonly used. Additionally, switching and routing delays are incorporated at system-level and parallel processing of algorithms as well as the strong dependence of the used HDL coding style turn hardware implementations into a challenging task [47]. Consequently, the designer requires extensive experience and knowledge for successfully carrying out such hardware implementations. One possible implementation design flow, which was used for a DSP-module implementation within the scope of this thesis, is illustrated in Figure 4.3.

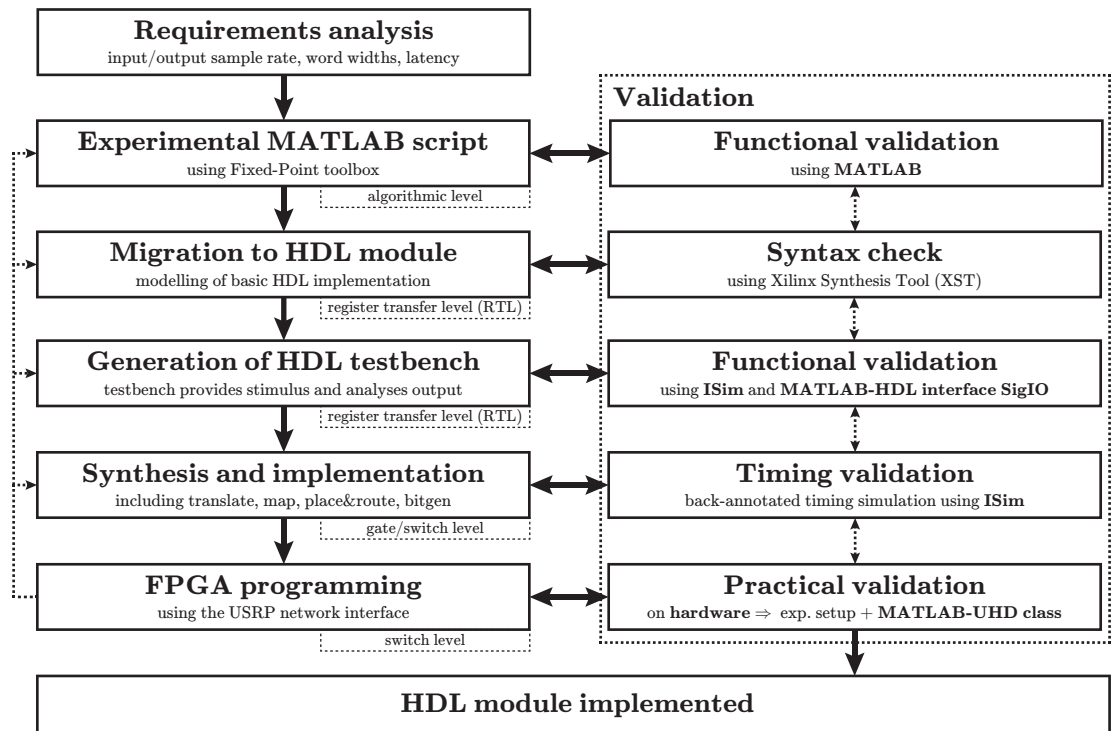


Figure 4.3: Implementation design flow

The general idea is to pursue a top-down approach, starting at the implementation of the algorithm at high-level (e.g. in MATLAB) continuing migrate to low-level HDL. The latter is then synthesised and mapped onto the specific FPGA architecture, and eventually ending in the programmed FPGA. Likewise, it can also be thought of as hierarchical abstraction levels of the implemented algorithm. Each abstraction level only represents the necessary information that is important to the corresponding level. While proceeding further in the implementation process, the level of abstraction decreases and the accuracy of the hardware model increases. Therefore, more details, such as timing information, are added to the description.

For example, the algorithmic level is made up of the top level describing the function of the algorithm using high-level constructs. In the RTL, digital circuits are modelled by basic operations and the transfer of data and control signals [27]. The synthesis tool translates the RTL code into a gate-level netlist containing the logical gates such as ANDs, ORs and XORs. Next, the implementation tools map the netlist onto the physical components of the FPGA, yielding the switch level of the implementation. Before proceeding to the next stage within the implementation design flow, simulations aid to check the outputs of the current implementation against the outputs of the prior stage, as indicated by the dashed pointers between the validation boxes in Figure 4.3. Once the simulation is successful, the implementation process is continued to the next lower level. However, a simulation may reveal problems that can possibly not be resolved in the current implementation stage. This would require going back multiple steps and starting from there again.

The individual processes, which are part of the design flow, are described in the following. The HDL implementations are solely focused on the HDL language Verilog since the existing firmware design is already based on it [48, 49].

### 4.2.1 Requirement Analysis

At the beginning of the implementation process, a requirement analysis has to be conducted. The requirements are strictly application-dependent and have to be individually investigated for each module. With regard to DSP-related implementations, parameters such as the input and output sample rate, the maximum latency, sample widths and possible resource limitations are identified to form the most relevant requirements. The sample rates dictate how fast the module needs to process the samples, whereas the latency defines a maximum tolerable processing delay. To account for the bit growth, which results after each arithmetic operation of fixed-point numbers, either rounding or truncation has to be performed. Therefore, the sample word widths, as well as the number representation and its fractional length, need to be specified.

### 4.2.2 Experimental MATLAB Script

Once these requirements are specified, an initial implementation of the algorithm can be carried out in a high-level environment (e.g. MATLAB). The implementation is thereby described on an algorithmic abstraction level [27]. The *Fixed-Point Designer* of MATLAB enables a setup of an experimental environment in order to alleviate the

overall HDL implementation process [50]. In this way, low-level-specific arithmetic operations, such as bit-shifts, bit truncations and rounding methods, can be implemented while still using high-level MATLAB constructs (e.g. loops) as well as convenient visualisation features. Furthermore, rich debugging functionalities represent a superior possibility to optimise the implementation with regard to a low-level description.

### 4.2.3 Migration to HDL Module

Next, the high-level MATLAB implementation is to be migrated to a low-level HDL module. All high-level constructs, such as loops or proprietary MATLAB functions, have to be converted into low-level descriptions. Basically, digital circuits can be described in HDL in both gate level and RTL. As being pursued for this implementation, the latter suits particularly well for synchronous designs, since the data is synchronously transferred between the registers [27]. Figure 4.4 illustrates an exemplary HDL module to implement.

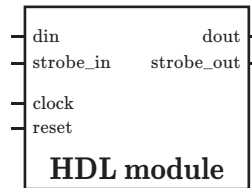


Figure 4.4: Exemplary HDL module

The module is provided with a `clock` signal as the clock input and a `reset` signal to reset all register ports within the module. The ports `din` and `dout` correspond to the input and output data buses. The `strobe_in` and `strobe_out` signal ports indicate whether the input and output is valid, respectively. A simple syntax check is performed by XST. However, no statement about the correct functionality of the implemented HDL module can be made at this point.

### 4.2.4 HDL Testbench

Proceeding in the implementation design flow, the HDL module must now be verified for its correct functionality. In order to do so, test signals, called stimuli, are excited at the input ports of the module and the resulting output signals are checked. Therefore, it can be investigated whether or not the implementation behaves as expected. This is realised by another, separated HDL module embracing the HDL module under test, denoted as the unit under test (UUT), as depicted in Figure 4.5. [51, 27]

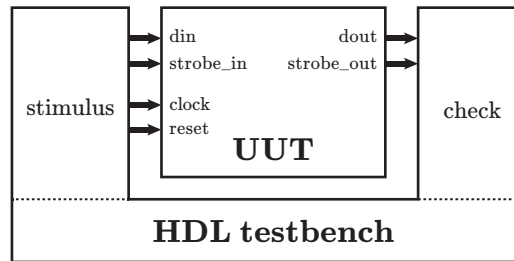


Figure 4.5: HDL testbench providing stimulus to the HDL module UUT

An HDL testbench instantiates the UUT and stimulates it with data test signals, as well as a clock and a reset signal, as demonstrated in Listing 4.1. Lines 4-5 show how a clock with a frequency of 100 MHz is generated in the testbench. Lines 13-19 instantiates the UUT *hdlmodule* and wires the interconnections within the design. The initial block in lines 21-30 is used specifically in testbenches only, which is not synthesizable due to certain statements (e.g. delays), and executes the enclosed code once [51]. Hereby, the UUT can be stimulated with an arbitrary input that is scheduled for certain instants of time, which are set by the `#` delay operator (line 23 and 25). The unit and the resolution of the simulation is declared by the *timescale* compiler directive in line 1 (time unit/resolution). Furthermore, the stimulus input can also be read from an ASCII file, as being established by the MATLAB-HDL interface explained in the following.

Listing 4.1: Simple HDL testbench

```

1  `timescale 1ns / 1ps
2  module hdlmodule_tb;
3
4  reg clock = 0;
5  always #5 clock = !clock;
6
7  reg reset;
8  reg [31:0] din;
9  wire [31:0] dout;
10 reg strobe_in;
11 wire strobe_out;
12
13 hdlmodule uut (
14     .clock(clock),
15     .reset(reset),
16     .din(din),
17     .strobe_in(strobe_in),
18     .dout(dout),
19     .strobe_out(strobe_out));
20
21 initial begin
22     $monitor("time %g: din=%b, dout=%b");

```



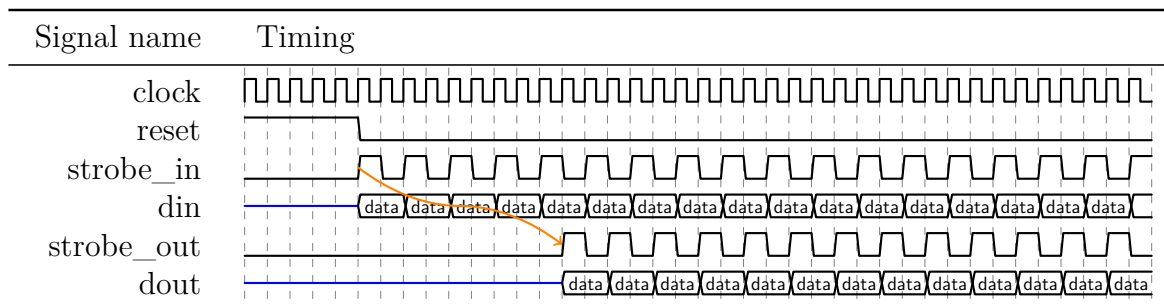
```

23     #5
24     reset = 1;
25     #50
26     reset = 0;
27     din = 32'b01010101010101010101010101010101;
28     strobe_in = 1;
29     ...
30 end
31 endmodule

```

System tasks, such as *\$monitor* (line 22) or *\$display*, are able to display the value of a signal. *\$fwrite* writes the value to a file [51].

Once the testbench is set up, functional, or behavioural, simulations can be run using an HDL simulator such as *ISim*, which is integrated in the *ISE Design Suite* [52]. Before running a simulation, the HDL source files including the testbench have to be parsed by *vlogcomp/vhpcomp* (for either Verilog or VHDL) in order to generate binary representations. Then, *fuse* creates a simulation executable by performing a static elaboration and generation of the object code [52]. When now calling the generated simulation executable, the graphical user interface (GUI) of *ISim* (same holds for other HDL simulators) illustrates the waveforms modelled by the HDL testbench, as shown in Timing diagram 4.1. This way the timing behaviour of the HDL module can be analysed assuming static delays for both components and interconnections.



Timing diagram 4.1: *bsd\_filter* module

In this case, a clock with a frequency of 100 MHz was generated and the reset signal is asserted for the first 5 clock cycles. Thereafter, the module is stimulated with data presented at port **din** in every second cycle, which is signalled by the **strobe\_in** signal. The processing latency of the module accounts for 9 clock cycles as it can be observed by the **strobe\_out** signal, which indicates the output **dout** to be ready.

### 4.2.5 MATLAB-HDL Interface SigIO

With the aid of the HDL testbench, it can be observed whether the implemented HDL module behaves as desired. However, as the content of the data samples has not yet been checked, the correct function of the module still has to be validated. Hence, a MATLAB-HDL interface *SigIO* is developed for performing advanced simulation methodologies, as depicted in Figure 4.6.

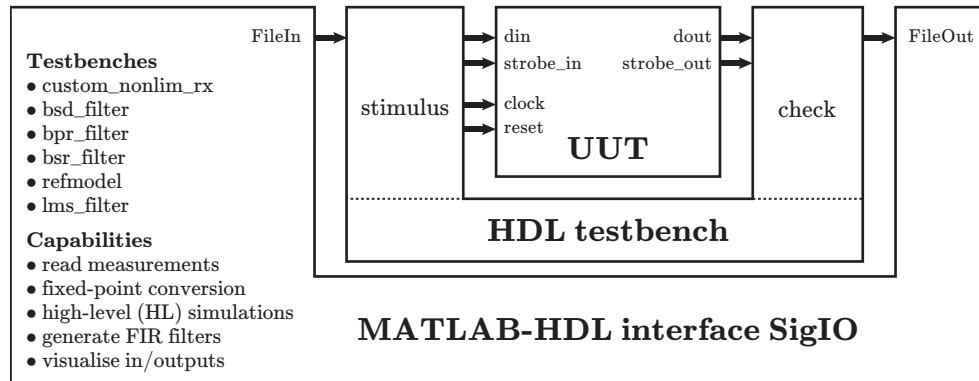


Figure 4.6: MATLAB-HDL interface interacting with the HDL testbench of Figure 4.5

Basically, the idea is to feed the HDL testbench with data that was generated by MATLAB in order to reproduce expected results. Therefore, the interaction of both MATLAB and the HDL testbench is based on a file I/O interface. It can be selected between the testbenches `custom_nonlim_rx`, `bsd_filter`, `bpr_filter`, `bsr_filter`, `refmodel` and `lms_filter`, corresponding to the sub-modules of NONLIM. *SigIO* is a MATLAB class with several associated properties in order to form a generic file I/O interface to the HDL testbench. The file I/O methodology of the *SigIO* class can be summarised as follows:

1. Generation of stimulus data (artificially or read by measurements)
2. Writing of stimulus data to an ASCII file
3. Compilation of Verilog source files (*fuse*)
4. Performing HDL simulation and writing output data back to ASCII file
5. Reading back the HDL testbench output
6. Visualisation of output

At first, the stimulus data is generated either by creating it artificially or by reading

data of real measurements that were conducted using the target architecture at hand. The former benefits from the advantage of being flexibly composable of various signal components, the latter provides a more realistic simulation environment that can be applied to either the entire NONLIM module or the individual sub-modules, which are to be implemented.

Generally, numbers in MATLAB contain 64 bit and are stored using the double-precision floating point format according to the IEEE Standard 754 for double precision [20]. Since the number representation in the existing firmware architecture is a fixed-point 2C number with 16 bit, *SigIO* has to perform a fixed-point conversion. This is done with the aid of the *Fixed-Point Designer*, also provided by Mathworks, that enables to set fixed-point properties such as word length and rounding mode and to perform bit-true operations [50]. The fixed-point-converted, binary stimulus data is then written to an ASCII file, where the rows correspond to samples in time and two columns (separated with a space) reflect the I/Q components, respectively. Before the compilation of the Verilog sources files can take place, the HDL testbench file needs to be extended so that it reads and writes the I/O data to the I/O files, which interface to the *SigIO* class. According to the decimated input sample rate of 12.5 MHz, the samples come in every 8 clock cycle. Hence, the HDL testbench reads and writes the rows (corresponding to I/Q pairs) at the same rate of 12.5 MHz from the input and output file, respectively. After the Verilog source files of the UUT, the HDL testbench, as well as the required simulation libraries [33] are compiled, the HDL simulation is performed. The simulation is started by calling the created simulation executable. Once finished, the binary 2C output of the UUT is stored by the HDL testbench in the respective output file.

Next, the I/Q data is read and optionally converted back to the double-precision floating point representation. At this point, the conversion is not essential, since most of the MATLAB function are capable of processing also fixed-point data types. Finally, the simulation output is visualised using plotting functions of MATLAB and compared to the expected results. Another feature of *SigIO* constitutes the generation of the FIR filter coefficients, using the *firpm* function of MATLAB's Signal Processing Toolbox [53]. Filter parameters, such as the stop- and the passband with their respective attenuations, are defined in the *SigIO* class. The calculated filter coefficients are stored in a *.coe* file that is then read from the *CORE Generator* GUI in order to generate a filter module. Furthermore, the *SigIO* class enables to perform HL simulations to simulate the selected testbench using MATLAB's HL functions. In this way, the simulation outputs of both the HDL and HL simulation can be compared to each other.

If both results yield to be equal, the function of the implemented HDL module is shown to be correct, leading to the next stage within the implementation design flow.

### 4.2.6 Synthesis and Implementation

Now, the synthesis and the implementation of the design can take place, following the procedure described in Section 2.6.2. The implementation tool *NetGen* creates a simulation model that is based on the timing information obtained after mapping and routing the implemented HDL module into the whole design [33]. Performing the so-called back-annotated timing simulation is similar to the behavioural simulation. *Fuse* takes the HDL testbench of the behavioural simulation, the Verilog files of the simulation model and the SDF file with the timing information for compiling the simulation executable. The timing simulation is started by invoking the generated executable. Utilising either the GUI of *ISim* or the *SigIO* interface, the simulation outputs of the timing simulation are now checked against the outputs of the former behavioural HDL simulation, where only fixed delays were assumed. If they are the same, the mapped and routed HDL module is validated to function correctly in terms of timing and the programming of the FPGA can follow. Otherwise, returning to a prior implementation stage is required, e.g. migration to HDL module, within the implementation design flow in order to revise the HDL code.

### 4.2.7 FPGA Programming

After having successfully synthesised, implemented and validated the HDL module by performing a timing simulation, the tool *BitGen* [33] generates the bitstream file to be programmed onto the USRP N210 with the *netburner* via the network interface [54]. Upon programming the device, a practical verification is conducted in order to validate the implementation on hardware. For example, an arbitrary signal generator inputs a signal to the USRP, where the received signal is processed by the DDC chain and the implemented module. The resulting signal is sent to the host PC and then further processed. This is established using a MATLAB-UHD interface, developed by the Electronic Measurement Research Lab of the Ilmenau University of Technology (Germany), managing the transfer of the I/Q samples between the USRP device, the UHD interface and MATLAB. Thus, the practical functionality can be validated by checking the received output against what would be expected of a known input signal. More detailed explanations of the experimental setup and the measurements being carried out are outlined in Chapter 5.

## 4.3 Modules Implementation

Below, the HDL implementation of the individual sub-modules of NONLIM is described, following the modularised structure of Figure 3.2.

### 4.3.1 Implementation Constraints

The objective of the NONLIM algorithm is to cleanse the received signal spectrum of any IMD products induced by any interferer, which is a challenging task in practice. Thus, several implementation constraints are set in order to simplify the design and to facilitate the implementation. First, the FIR filters are identified to be the main components that would be potentially needed to adapt to a changing scenario. However, changing an FIR filter requires filter coefficients to be recalculated and the algorithm would be too complex to be integrated directly in the hardware. Therefore, with regard to the initial hardware implementation of NONLIM, the assumption is made of the most basic two-tone scenario, as described in Chapter 3. Thereby, the interferer band is set to exhibit a certain BW with a fixed position, significantly easing the implementation of the FIR filters. According to the previously selected decimation rate of 8 and the resulting BW of 12.5 MHz, the centre of the interferer band is chosen to be 3 MHz off the centre of the BB. When setting the BW of the interferer band to approximately 600 kHz, the two tones are likewise spaced with approximately 600 kHz to each other and therefore located at the edges of the interferer band at 2.7 MHz and 3.3 MHz. Thus the bandstop and bandpass filters need to be designed accordingly. Further filter-related constraints are discussed in Section 4.3.3.

### 4.3.2 Integration into Existing Firmware

In order to implement the NONLIM algorithm, an analysis of the firmware of the USRP N210 has revealed a convenient place-holder, where custom logic can be placed within the DDC chain of the existing FPGA firmware. This is established by the `./sdr_lib/dsp_rx_glue.v` module embracing the existing DDC chain, as illustrated in Section 4.1.5. The custom DSP module is activated by the Verilog macro `RX_DSP0_MODULE` that needs to be set before the synthesis of the design. Furthermore, the Verilog sources included in the custom module need to be added to the design so that they are also included in the synthesis and implementation process. Modifying the parameters `CUSTOM_SRC` and `CUSTOM_DEFS` in the makefile `./top/N2x0/-Make.N210R4` allows to perform the design synthesis and implementation via command

shell. This facilitates the process of activating the module and including the Verilog source files located in path `./custom/` into the project, as shown in Listing 4.2.

Listing 4.2: Modifications in file `Makefile.N210R4` in order to add a custom module

```

12 CUSTOM_SRCS = $(abspath $(addprefix $(BASE_DIR)/../custom/, \
13 custom_nonlim_rx.v \
14 custom_nonlim_rx_tb.v \
15 ))
16 CUSTOM_DEFS = RX_DSP0_MODULE=custom_nonlim_rx

```

Figure 4.7 shows the custom NONLIM block that is interconnected with the respective elements of the DDC chain located in the digital back-end. After the received analogue signal is digitised by the ADCs, with a resolution of 14 bit, and the DC offset and I/Q imbalance digitally corrected as far as possible [55]. The I/Q samples, which have grown after the arithmetic operations of the corrections to 24 bit, reach the custom module in every clock cycle. This corresponds to the full ADC sample rate, or equally the FPGA master clock, of 100 MHz. Since the NONLIM block should be placed after the DDC chain, the ports `frontend_i/q` are just bypassed to `ddc_in_i/q` using a continuous assignment. Hereafter, the samples are decimated by the DDC chain to a sample rate of 12.5 MHz, in accordance with the chosen decimation rate of 8. Afterwards, they are then rounded to 16-bit values and inserted as concatenated I/Q pairs into port `ddc_out_sample` of the custom module block, with `ddc_out_strobe` being asserted.

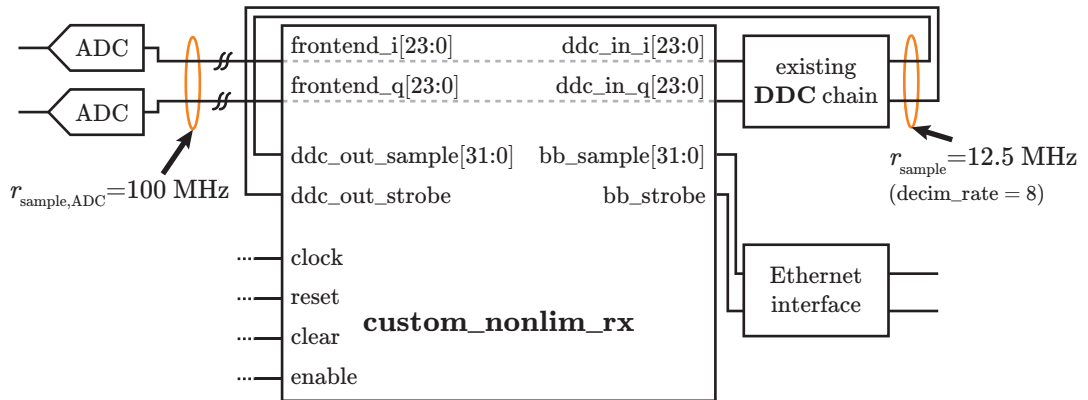


Figure 4.7: Custom module being integrated into the DDC chain

From this point, the NONLIM algorithm starts to process the samples. Once finished, the processed samples are output on the port `bb_sample` with the strobe signal `bb_strobe` set high.

### 4.3.3 FIR Filters

There are three instances of FIR filters employed by the NONLIM algorithm, denoted as *bsd\_filter*, *bpr\_filter* and *bsr\_filter*. The design of the filters is integrated into the *SigIO* class, where the filter coefficients are generated using the *firpm* function with the Parks-McClellan algorithm, which is part of MATLAB's Signal Processing Toolbox [21]. In order to design the filters, the specifications of the stop- and the passband and the corresponding attenuation factors need to be defined, as depicted for an arbitrary bandstop filter in Figure 4.8.

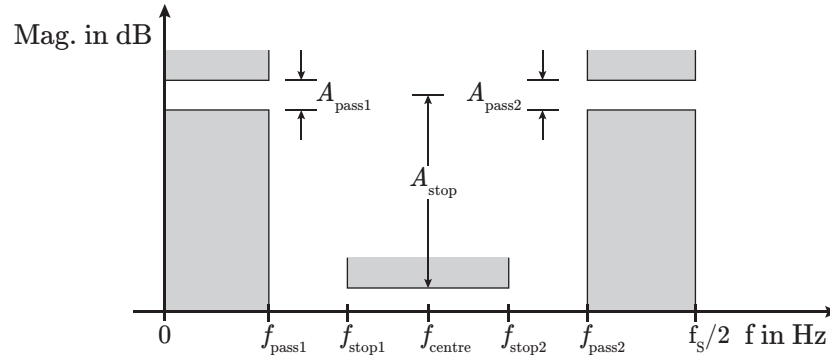


Figure 4.8: Filter specifications of a bandstop filter, taken from [20]

The frequency parameters  $f_{\text{pass1/2}}$  and  $f_{\text{stop1/2}}$  define the range of both the stop- and the passband,  $A_{\text{pass1/2}}$  the ripples that are allowed within the passband and  $A_{\text{stop}}$  the main stopband attenuation factor. Hence, the Parks-McClellan algorithm tries to fit the filter's frequency response into these specifications using the Remez exchange algorithm and the Chebyshev approximation theory [21]. In general, the filter order  $M$  strictly depends on how tight the filter specifications are defined. That is, increasing the stopband attenuation  $A_{\text{stop}}$ , as well as decreasing the transition width  $|f_{\text{pass1/2}} - f_{\text{stop1/2}}|$  and the passband ripples  $A_{\text{pass1}}$  yields an increased filter order  $M$ . This results in an increased amount of required hardware resources in terms of the implementation. Thus, a trade-off between sufficient filter characteristics and a reasonable amount of resource utilisation has to be found. The shape of the filter response is assumed to be symmetric around  $f_{\text{centre}}$ , it can be defined

$$\begin{aligned} f_{\text{cutoff1}} &= f_{\text{stop2}} - f_{\text{centre}} = f_{\text{centre}} - f_{\text{stop1}} \\ f_{\text{cutoff2}} &= f_{\text{pass2}} - f_{\text{centre}} = f_{\text{centre}} - f_{\text{pass1}}. \end{aligned}$$

filter instance	BW	$f_{\text{cutoff1}}$	$f_{\text{cutoff2}}$	$A_{\text{stop}}/A_{\text{pass1/2}}$	filter order M
<i>bsd_filter</i>	640 kHz	320 kHz	600 kHz	85/0.2 dB	161 ( $M_{\text{bsd}}$ )
<i>bpr_filter</i>				40/0.2 dB	95 ( $M_{\text{bpr}}$ )
<i>bsr_filter</i>					95 ( $M_{\text{bsr}}$ )

Table 4.2: FIR filter specifications with  $f_{\text{centre}} = 3$  MHz

Table 4.2 lists the filter design parameters that are being used for the NONLIM implementation. The pass- and stopband of the bandstop and bandpass filters are merely swapped so they exhibit inverse characteristics. A stopband attenuation of 85 dB is chosen for the *bsd\_filter* in order to get a sophisticated mitigation performance, as it will be shown in Chapter 5.

The design parameters that have just been elaborated are used to generate the filter coefficients using the Parks-McClellan algorithm. The coefficients are represented in double-precision floating point numbers and must therefore be converted into a fixed-point format. This results in an altered filter response. Given a fixed-point format, the maximum word length of a coefficient is set to 18 bit, which is dictated by the embedded multipliers residing in the DSP48A blocks that are commonly used for filter implementations in Spartan-3A architectures. It is noteworthy that fixed integer and fractional length for all filter coefficient sets might not be the best choice for the following reason. For example, when assuming a Q3.14 signed fixed-point number format for representing the coefficients and the magnitude of all filter coefficients being less than one, the three integer bit positions of the Q3.14 number would constitute redundant information. Therefore, a Q17 format would yield a more precise representation. Hence, the precision of the coefficients can always be optimised so it follows a best precision fractional length of the individual filter coefficient sets [24]. Figure 4.9 compares the magnitude and phase responses of the *bsd\_filter* resulting from the quantisation of the filter coefficients using a Q3.14 and a Q17 number format with the responses using a double-precision representation. Accordingly, the Q17 seems to be the number format yielding the best precision fractional length.

It can be observed that both the magnitude and phases do not change significantly in the passband after quantisation, however, the magnitude response exhibits a few irregular peaks, which lowers the stopband attenuation by approximately 5 dB to around 80 dB, which is still tolerable. The stopband attenuation degrades even further to about 68 dB if the number format of the coefficients is set to a non-optimal value, e.g. Q3.14, decreasing the precision of the filter coefficients. Hence, the usage of the best precision fractional length is always preferred.



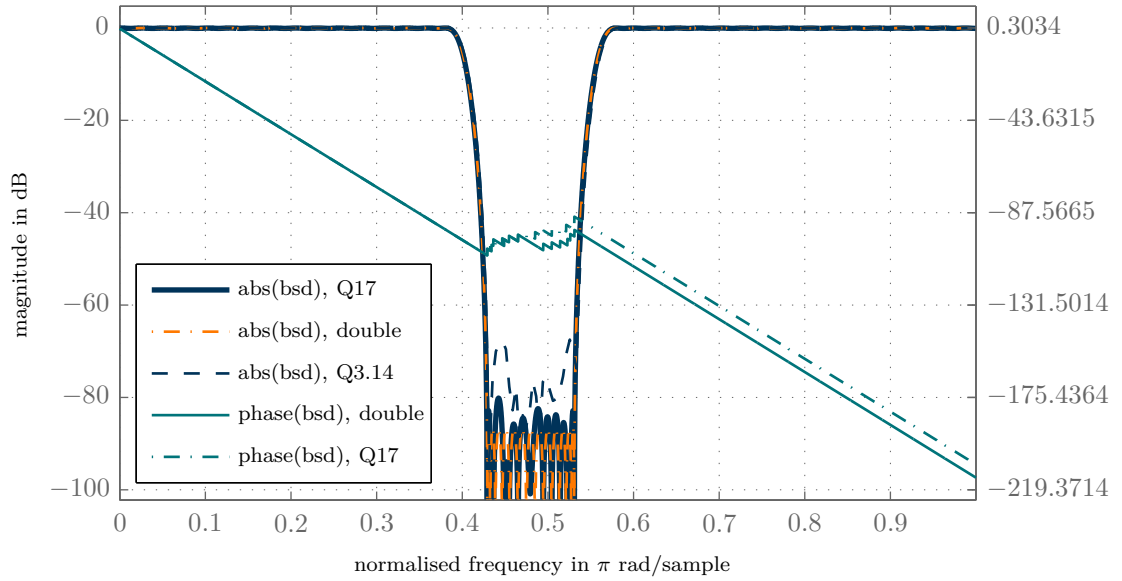


Figure 4.9: Magnitude and phase response of the quantised FIR filter *bsd\_filter*

Hereafter, the filter coefficients are exported to the Xilinx-specific coefficient file format `.coe` [24], which allows to easily load the coefficients into the *CORE Generator* GUI of the *FIR compiler* [24]. Several parameters such as the input sample rate, the clock rate, the word and fractional widths of the input/output samples and coefficients and the filter architecture have to be configured in the GUI of the FIR compiler. The GUI also allows to perform rounding of the output, however, a manual rounding within the HDL code is chosen due to the advantage of an increased control.

When generating the filter, two main filter architectures can be selected: MAC and Distributed Arithmetic (DA). The former represents the most conventional FIR architecture, as it is based on multiplications and accumulations. Thereby, multiple DSP48A slices are chained together in order to form an area-efficient and a high performance realisation of FIR filters [24]. Figure 4.10 shows the chosen symmetric systolic FIR filter structure, where the clock-triggered FFs (configured by the OPMODE) delay the data samples in order to fully utilise the architecture of the DSP48A slices [56].

When implementing filters that exhibit a symmetric coefficient structure, the area-efficiency can further be increased by adding the respective samples of the first and second half and jointly multiplying them by the corresponding coefficients. This saves half of the amount of the required DSP48A blocks. After the multiplication operations, the products are accumulated using the embedded post-adder chain [56].

Another fundamentally different filter architecture is represented by the DA. The key of the DA structure is that the multiplications are substituted by bit-shifts, addi-

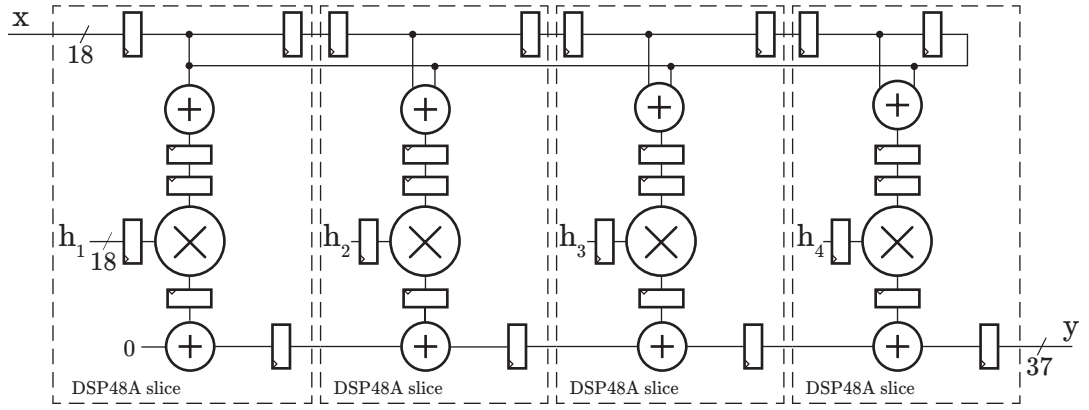


Figure 4.10: Simplified diagram of a symmetric systolic FIR filter implementation, taken from [56]

tions, subtractions, and LUTs. Thereby, equivalent multiplication operations are not processed as a whole but split up into bit-wise sums of partial products. These partial products correspond to particular combinations of the input sample and the coefficient. Since the coefficient is constant, the possible combinations can be stored in a LUT. Hence, the individual bits of the input samples address the inputs of the LUT in order to retrieve the partial sums, which are bit-shifted and accumulated accordingly. More detailed information can be found in the literature [57]. The usage of the DA filter structure is suited particularly well when multipliers have to be saved. However, more CLBs are utilised in return. With regard to the NONLIM implementation, the DA was used for the *bpr\_filter* realisation only in order to find a balance between the utilisation of DSP48A blocks and FPGA fabric.

Performing a filtering process of communication systems requires that both I/Q components are processed. Instead of generating two individual IP core instances, the *FIR Compiler* provides two kinds of processing multiple streams: a multi-channel type that uses a time division multiplex (TDM) sharing scheme and a parallel data path type processing the input streams in parallel [24]. The former accepts both I/Q channels alternately on the same input port `din` and the results are output in the same way. These are indicated by two output ports `chan_in` and `chan_out`. The latter features two dedicated input and output ports `din_1/2` and `dout_1/2` for both I/Q channels. The choice between either is made by means of the utilised hardware resources of both realisations. When considering *bsd\_filter* with a symmetric coefficient structure, the number of DSP48A slices required can be computed as follows. Given a clock of 100 MHz and an input sample rate of 12.5 MHz yields 8 clock cycles per sample and a so-called hardware oversampling rate of 8 cycles/2 channels = 4 cycles/input sample

[24]. That is, each input sample has to be processed within 4 cycles. Exploiting the symmetric filter structure, the processing of the 161 filter taps gives a total resource usage of  $81 \text{ symmetric coefficients} / 4 \frac{\text{cycles}}{\text{input sample}} = 21 \text{ DSP48A slices}$  [24]. A Parallel Data Path realisation, where either channel can take the full 8 cycles for processing each input sample, yields a resource usage of  $81 \text{ symmetric coefficients} / 8 \frac{\text{cycles}}{\text{input sample}} = 22 \text{ DSP48A slices}$  [24]. Consequently, a multi-channel realisation represents a slightly more efficient filter implementation and is therefore used for the *bsd\_filter* and *bsr\_filter* implementations.

Another important point to consider is that common filter design functions of MATLAB, such as the *firpm*, generate filters with real filter coefficients for filtering real signal, showing a symmetric magnitude response. If those real filter coefficients are applied to both I/Q branches, the filter response exhibits a symmetric filter characteristic around DC in BB. In some cases, this might be sufficient as only one coefficient set has to be applied to both I/Q components and therefore hardware resources are saved. However, sometimes a strictly non-symmetric filter characteristic might be desired, which requires the performance of complex-valued filtering. Complex filter coefficients can be computed by designing a filter prototype that is centred around DC and then shifted by performing a complex frequency shift transformation. This is achieved by multiplying the filter coefficients by  $e^{j2\pi f_{\text{centre}}n}$ , where  $f_{\text{centre}}$  is the frequency to shift the prototype filter to and  $n$  ranging from 0 to the number of filter taps  $N$ , yielding complex filter coefficients. Multiplying a complex coefficient  $c$  by a complex input sample  $x$  yields

$$\begin{aligned} c \cdot x &= (c_I + jc_Q)(x_I + jx_Q) \\ &= (c_I x_I - c_Q x_Q) + j(c_I x_Q + c_Q x_I). \end{aligned} \quad (4.1)$$

That is, complex filtering includes the multiplications and cross-multiplications of the real and imaginary components of an complex input sample  $x$  and coefficient  $c$ . The resulting products are added or subtracted accordingly, forming the I/Q components of the filter output. In terms of the filter implementation, the amount of required hardware resources is doubled as now four channels instead of two have to be processed and additional accumulation logic is needed. The *bpr\_filter* represents the only filter realisation of the NONLIM algorithm where a complex filtering stage is crucial for the IMD mitigation. Implementing the *bpr\_filter*, a multi-channel filter implementation with two channels is created with the *FIR Compiler*, using real and imaginary filter coefficient sets, respectively. A DA filter architecture is chosen for the *bpr\_filter* to

find a sound balance between the utilisation of DSP48A slices and CLBs with regard to the NONLIM implementation.

Once the filter instances are generated using the previously determined configurations, the *CORE Generator* creates a functional verification model, which enables to simulate the created IP core, and an instantiation template. Hereby, the port connections, as depicted in Figure 4.11, must be changed to the corresponding signal names of the HDL module [24].

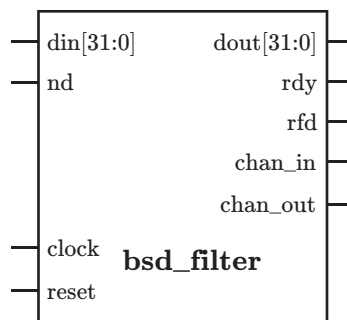
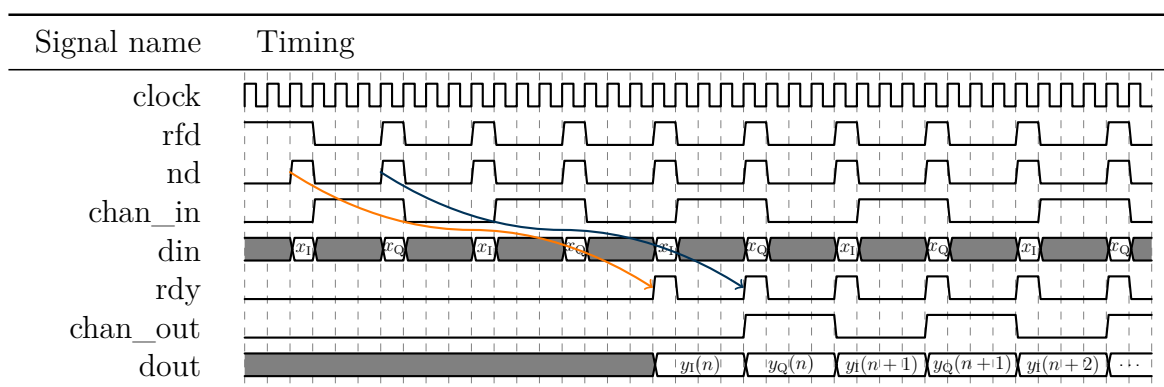


Figure 4.11: Generated *bsd\_filter* block to be instantiated within the HDL design

Apart from the I/O ports `din` and `dout`, the filter module with the selected configuration employs the ports `chan_in` and `chan_out`, which indicate the active channels of the I/O streams that is asserted on `din` and `dout`. Moreover, additional handshake control signals `nd`, `rfd` and `rdy` ensure a proper data flow to the filter module. The Timing diagram 4.2 shows the filter timing of an exemplary I/Q multi-channel implementation, which was created by the *FIR Compiler*.



Timing diagram 4.2: Exemplary I/Q multi-channel filter implementation created by the *FIR compiler*, taken from [24]

The filter core indicates by the active high output signal `rfd`, whether it is ready or not for accepting data. When being asserted, the active high input signal `nd` is set high.

This signals the core that a data sample  $x_{I/Q}$ , provided on `din`, is valid. Additionally, the module output signal `chan_in` indicates, if either the I- or the Q-channel is to be inserted on `din`. Once the module has finished processing of an individual sample, `rdy` is set high and the output sample  $y_{I/Q}(n)$  is active on port `dout`. The port `chan_out` indicates if either the I- or Q-channel is present on `dout`, respectively. `dout` is registered internally so that it can be accessed until the subsequent output sample is written. In terms of the exemplary filter implementation, shown in Timing diagram 4.2, the cycle latency for processing a sample  $x_{I/Q}(n)$  and retrieval of the output  $y_{I/Q}(n)$  accounts for 16 cycles. This is indicated by the orange and blue pointers corresponding the two channels.

Since the output rounding is disabled when generating the filter module, the rounding must be performed by hand within the HDL code in order to maintain a constant word length of the samples. This is done by using the *round* module of the existing FPGA firmware. The module allows for a performance of either a simple bit truncation, symmetric rounding to zero or non-symmetric rounding towards positive [24]. The latter method is chosen, according to the usage in the existing firmware. The rounding within HDL module implies the advantage that the I/Q word length as well as the bits to round can be chosen freely. Therefore, it is more flexible compared to the integrated rounding of the generated filter module, whereby only the topmost bits are rounded. The rounding operation takes one clock cycle of duration and the amount of utilised resources for rounding a 36-bit value to 16-bit accounts for 12 CLBs.

#### 4.3.4 Reference Model

In the next step, the implementation of the reference model block, which generates the reference non-linearity in the reference branch of NONLIM, is presented. The implementation of the reference model is basically described by (3.1). The factor of  $3a_3$  was left out as  $a_3$  is not known in practice in order to save another multiplier for both I- and Q-channels. Since the LMS filter scales the reference distortions anyway, this simplification has no effect on the mitigation performance. Consequently, (3.1) is reduced to

$$\tilde{u}^2 \tilde{u}^* = \dots = (u_1^3 + u_1 u_Q^2) + j(u_Q^3 + u_1^2 u_Q), \quad (4.2)$$

which is eventually implemented. Figure 4.12 shows a block diagram of the implementation structure of the *refmodel* block.

The arithmetic operations are conducted sequentially, whereby the individual mul-

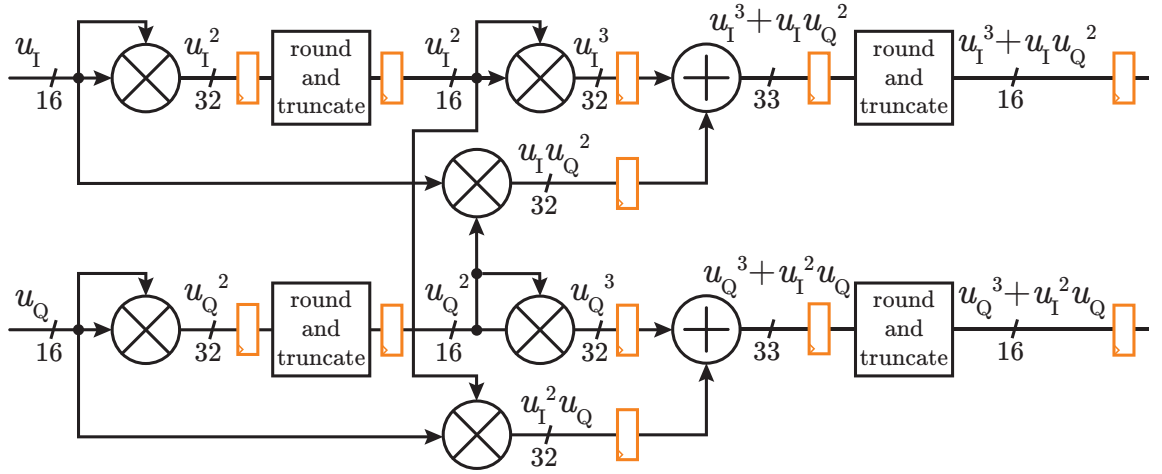


Figure 4.12: Pipelined implementation structure of the *refmodel* block

tiplications are separated and then added accordingly. Moreover, round-and-truncate blocks perform rounding (non-symmetric to positive) and bit truncations of the LSBs in order to account for bit growth resulting after each arithmetic operation.

Performing multiple operations within one clock cycle would result in the design being slowed down, as the maximum achievable clock is the reciprocal of the duration of the most critical path (i.e. the slowest path). The maximum duration of the logic path is commonly formulated by a *PERIOD* constraint defined in the UCF, which is input into the design implementation process. If such a *PERIOD* constraint fails during the implementation process, the respective implementation tool reports an error as a timing constraint violation. Such timing errors are often caused by paths that comprehend lengthy routing delays within the design or involve too many levels of combinational logic. The design would fail to meet the maximum frequency of 100 MHz if all multiplications and additions are combined in one cycle. One way to account for this is to insert additional pipeline registers between the processing blocks to divide the computation into multiple cycles [58]. In fact, this results in an increased processing latency of the algorithm but the overall maximum achievable frequency of the design is maintained. Due to the insertion of the pipeline register, which are inserted into the *refmodel* implementation structure, the total processing latency of the *refmodel* block accounts for 10 clock cycles.

### 4.3.5 Delay Blocks

Referring to Figure 3.2, two delay blocks are employed in the NONLIM algorithm: *d\_delay* and *r\_delay*. Both blocks are implemented using a 32 bit  $\times$  512 bit FIFO,

which are generated with the *CORE Generator FIFO Generator* [59]. Thus, the FIFO is capable to store 512 32-bit wide I/Q samples utilising an embedded 18kbit block RAM. Apart from the I/O data ports, the generated IP core module also provides two active high input ports **rd\_en** and **wr\_en**, which signal a reading or writing operation, respectively. The read and write principle follows that of the first in, first out. When writing a sample, **wr\_en** is asserted and the data sample, which is presented on **din**, is stored in the FIFO. One cycle after **rd\_en** was set high, the data sample that was firstly stored in the FIFO is presented on the output port **dout** and subsequently removed from the FIFO. In this way, the FIFO is used as a flexible buffer for storing and reading data samples at certain instants of time.

The task of the *d\_delay* module is to eliminate the timing offset between the desired and the reference branch, which are caused by different processing chains. Therefore, the *lms\_filter* module is fed with two perfectly aligned signals. The timing offset comprises both the processing delay  $n_{\text{proc}}$  and the settling time delay  $n_{\text{settle}}$  induced by the FIR filters, as demonstrated in (3.2). The former is bypassed by coupling both branches of the corresponding modules with the aid of the *bsr\_filter* strobe signal of the reference branch. For example, the output strobe **bsr\_strobe** of the *bsr\_filter* block is wired with the **rd\_en** port of the *d\_delay* module so that the samples of both branches simultaneously reach the *lms\_filter* module. Hence, the pure processing delay is self-regulated by the NONLIM algorithm, whereby the remaining settling time delay  $n_{\text{settle}}$  still needs to be taken into account. The settling time for a symmetric filter of the order  $M$  amounts to  $M/2$ . Since both the desired and the reference branch exhibit a certain settling time, a consideration of the difference in settling time delays is significant to align both branches. This yields

$$\begin{aligned}
 n_{\text{d\_delay}} &= n_{\text{r,settle}} - n_{\text{d,settle}} = n_{\text{bpr,settle}} + n_{\text{bsr,settle}} - n_{\text{bsd,settle}} \\
 &= \left(\frac{M_{\text{bpr}} + 1}{2} + 1\right) + \left(\frac{M_{\text{bsr}} + 1}{2} + 1\right) - \left(\frac{M_{\text{bsd}} + 1}{2} + 1\right) \\
 &= 49 + 49 - 82 = 16,
 \end{aligned} \tag{4.3}$$

where the units of  $n$  are given in samples. Hence, the reference is the desired branch 16 samples ahead in terms of giving a valid output. In order to establish an alignment of both, the first 16 **bsr\_strobe** signals are skipped, corresponding to dropping the first 16 samples of the reference branch. This is realised by a counter that counts up to  $n_{\text{d\_delay}} = 16$  on each **bsr\_strobe**. Once the 16 samples are elapsed, the **bsr\_strobe** is activated to be directly switched through to the *d\_delay* module so that both  $\tilde{d}[n]$

and  $\tilde{r}[n]$  can be presented to the *lms\_filter* module. Consequently, both branches are perfectly aligned to each other.

The second delay block, which should be implemented in NONLIM, is the *r\_delay* module. According to Figure 3.2, the interferer band  $\tilde{u}[n]$  is isolated by the *bpr\_filter* from the rest of the spectrum and is not contained any more in the output of the NONLIM block. This is reasonable when a single band of interest is considered. However, it is not the desired behaviour when the BB should be only cleaned of non-linear distortions, e.g. for the spectrum sensing in the context of CR. In order to retrieve the original signal  $\tilde{x}[n]$  with the removed non-linear distortions, the interferer is added back to the LMS output  $\tilde{e}[n]$ , taking any timing offsets induced by the reference branch into account. Following the same proceedings as for the *d\_delay* block, the *lms\_strobe* tackles the processing delay and the FIFO block *r\_delay* accounts for the settling time delay. The time settling delay  $n_{r\_delay}$  equals the settling time of the *bsr\_filter* module yielding  $n_{r\_delay} = \frac{M_{bsr}+1}{2} + 1 = 49$  samples. Likewise, a counter is incremented on each *lms\_strobe\_out* signal. After 49 output samples of the LMS filter, the *lms\_out\_strobe* triggers the samples  $\tilde{u}[n]$ , which are buffered in *r\_delay*, to be read out and added back to the LMS filter output  $\tilde{e}[n]$ . It is important to note that, in contrast to the *d\_delay* module, no output samples  $\tilde{e}[n]$  are dropped before the counter has elapsed. Instead, the addition of the interferer signal is simply postponed by the settling time delay of *bsr\_filter*.

### 4.3.6 LMS Adaptive Filter

The task of the LMS filter is to adjust the reference distortions  $\tilde{r}[n]$  that were artificially generated in the reference branch to the actual distortions contained in signal  $\tilde{d}[n]$ . The method of adaptation can be divided into the two sections. Firstly the FIR filter, having its adaptive coefficients, and secondly the coefficient update part, as illustrated in Figure 4.13. Since both I/Q components are processed separately, the computation procedure of both components is the same and therefore the illustration is restricted to the I-component only.

#### Adaptive FIR Filter

The task of the FIR filter is to scale the reference distortions  $\tilde{r}[n]$  so that they match the actual ones of  $\tilde{d}[n]$ . The filter coefficients  $b_{i,I/Q}[n]$  are thereby adapted at each snapshot  $n$ . The filter is realised using  $L$  embedded, parallel multipliers, taking the actual and the  $L - 1$  past samples of  $\tilde{r}[n]$  and the adaptive filter coefficients  $b_{i,I/Q}[n]$  of



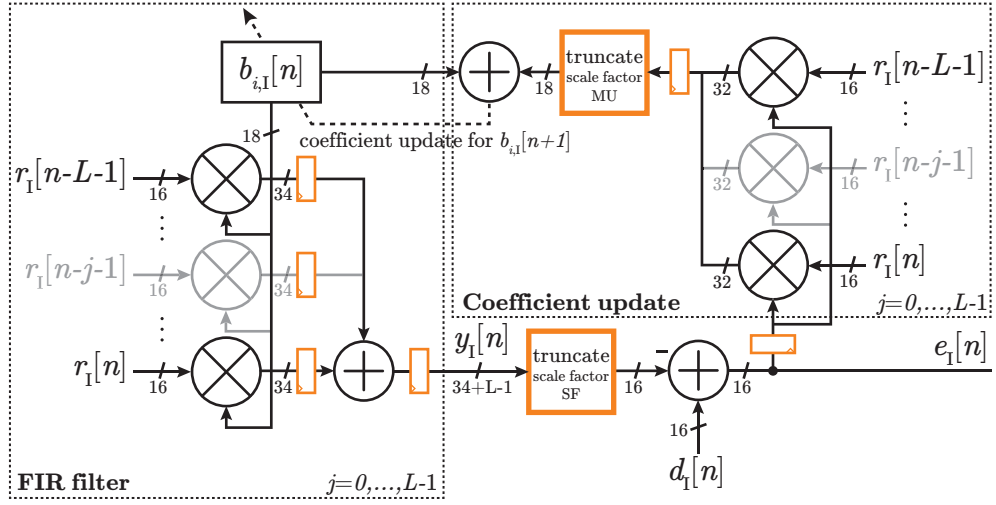


Figure 4.13: Implementation structure of the LMS filter (I-component only)

the current time instant  $n$  as its operands, as well as a subsequent adder. The usage of the parallel multipliers allows for a calculation of the partial products within one cycle. The subsequent accumulation of the partial products is divided into two cycles, since the addition of multiple 34-bit numbers involves several levels of logic in hardware, what might cause difficulties in meeting the timing constraints of the design implementation process. After each summation, the bit width potentially grows by one bit, yielding a word length of in total  $34 + L - 1$ . The output of the FIR filter  $y_{I/Q}[n]$  needs to be mapped to a 16-bit wide result so that it can be subtracted from  $d_{I/Q}[n]$ , forming the output of the LMS filter  $e_{I/Q}[n]$ . The mapping is performed by an arithmetic right shift of  $y_{I/Q}[n]$  by a given number of bit positions  $SF$ , corresponding to a division by integers of powers of two. The arithmetic shift does not involve any rounding since it is assumed that the adaptive algorithm compensates for the incorporated bias that results from the bit-truncation of  $y_{I/Q}[n]$ .

### Coefficient Update

The coefficient update part of the LMS filter, depicted on the right-hand side of Figure 4.13, achieves the adaptation of the filter coefficients, implementing the coefficient update equation

$$\mathbf{b}[n+1] = \mathbf{b}[n] + \mu e[n] \mathbf{r}[n], \quad (4.4)$$

where  $\mathbf{b}[n] = [b_i, \dots, b_L]^T$  denotes the filter coefficient set of the current snapshot  $n$  [22]. Hence,  $L$  parallel, embedded multipliers are utilised to carry out the multiplications

of the LMS filter output  $e_{I/Q}[n]$  and the reference distortion signal samples  $\tilde{r}$  of the actual and the  $L - 1$  past samples. The resulting  $L$  products are bit-truncated and arithmetically right shifted by a given number of bit positions  $MU$ . Therefore, the results are kept within the given 16-bit signed word range. The multiplication of the step size  $\mu$  is hereby included and approximated by a simple bit-shift operation, as presented in [11]. Thereafter, the truncated partial products are added accordingly to the respective filter coefficients  $b_i$  of the current snapshot  $n$ , yielding the filter coefficient set  $\mathbf{b}[n + 1]$  of the subsequent snapshot  $n + 1$ .

In order to maintain real-time processing, the coefficient update procedure needs to be finished before the subsequent samples  $r_{I/Q}[n + 1]$  and  $d_{I/Q}[n + 1]$  come in, i.e. within the time period of 8 cycles in regards to a chosen decimation rate of 8. This includes both the component delays as well as delays that are induced by pipeline stages (highlighted in orange in Figure 4.13).

The two bit-truncations, highlighted in orange in Figure 4.13, are identified to be two significant parameters in terms of the mitigation performance of the LMS filter implementation. Upon performing the bitshifts, the scaled values need to fit strictly within the target range  $-2^{15} < x < 2^{15} - 1$ , corresponding to the given 16-bit 2C word of each I/Q sample component. Otherwise, an overflow occurs causing catastrophic jumps in the waveform due to the wrap property of the 2C. In simulation, an overflow check is employed into the *SigIO* class, which reports an error message if the chosen  $SF$  or  $MU$  results in an overflow error. In hardware, the capabilities to employ an overflow check are limited. Nevertheless, possible overflows become apparent as abrupt jumps in the waveform or if the frequency representation of the signal does not look reasonable. That is, either  $SF$  or  $MU$  needs to be increased depending on whether the overflow has occurred after either the FIR filter or the coefficient update. In hardware, overflows must strictly be avoided by adjusting the  $SF$  or  $MU$  using the runtime reconfigurability capabilities of the NONLIM implementation, which is discussed at the end of this chapter. Hence, the two parameters  $MU$  and  $SF$  significantly contribute to the performance of the LMS filter implementation and must be chosen carefully.

### Analysis of LMS Filter Parameters

To analyse the convergence behaviour of the filter coefficients and the performance mitigation, simulations are carried out with variable LMS filter configurations. The HL simulations are performed with the *SigIO* class utilising a HL fixed-point implementation of the NONLIM algorithm. This offers greater possibilities to flexibly change the LMS filter parameters and to visualise and analyse the simulated results. In order

to imitate a more realistic simulation scenario, in which the NONLIM algorithm will operate when implemented on hardware, measurements are carried out with the USRP N210 using the WBX front-end and the stock firmware version 3.5.0. The measurement data is then loaded to the experimental MATLAB environment to run hardware-in-the-loop simulations of the NONLIM algorithm. For evaluating the performance of the respective LMS filter configurations, an analysis is performed, which investigates the mean squared error (MSE), the coefficient convergence behaviour and the PSD of the LMS filter output among variable combinations of the parameters  $L$ ,  $SF$  and  $MU$ . The MSE of a certain snapshot  $n$  takes the square of the absolute value of the complex LMS filter output  $\tilde{e}[n]$  and averages it over multiple realisations  $m$ , thus yielding

$$\text{MSE}[n] = \mathbb{E}\{|\tilde{e}[n]|^2\} = \mathbb{E}\{|e_{i,I}[n] + je_{i,Q}[n]|^2\} = \frac{1}{m} \sum_{i=1}^m |e_I[n] + je_Q[n]|^2. \quad (4.5)$$

The MSE allows for the analysis of the adaptation process over multiple iterations by calculating the instantaneous power of the LMS output signal. After a specific convergence time, the LMS filter has converged to a certain MSE level, referred to as the steady-state. Likewise, the characteristics of the filter coefficients are analysed in order to compare the convergence behaviour among different LMS filter configurations. The results of a series of measurements, which are carried out with the *SigIO* class, are shown in Figure 4.14. The first three rows of Figure 4.14 demonstrate the dependence of the parameters  $L$ ,  $MU$  and  $SF$ , respectively, whereby the MSE is illustrated on the left-hand side and the coefficient convergence behaviour of the coefficients is shown on the right-hand side column. 1000 iterations are performed, respectively, and the MSE and the PSDs are averaged over 15 realisations. In terms of the PSD, an FFT length of 1024 samples is used and a Hann window is applied [60]. When performing the simulations, it turns out that both  $SF$  and  $MU$  cannot be arbitrarily chosen since values that are too small result in an overflow of either the FIR filter output or the coefficient update process, which is reported by the *SigIO* class. With regard to the choice of parameters, a constraint can be deduced that the sum of  $SF$  and  $MU$  needs to be greater than or equal to 26 in order to avoid overflow, as already formulated in [11]. Besides, the values are determined within the scope of this series of measurements are not generally applicable but depend on the signal constellation at hand. Likewise, there is a dependence of the optimal parameters  $SF$  and  $MU$  for different filter orders  $L$  in order to make the LMS filter operating optimally. In Figure 4.14a and Figure 4.14b, exemplary LMS filter configurations are illustrated, comparing the convergence behaviour between different filter orders  $L$  with specific values for  $SF$  and

$MU$ , which are considered as the optimal ones. At the beginning of the adaptation process, all filter realisations exhibit a rising slope until approximately 80 iterations. This is due to the settling time of the FIR filters and corresponds to the filter with the longest settling period, which is the *bsd\_filter*, accounting for  $(M_{\text{bsd}} + 1)/2 = 81$ . From then on, the filter configurations with 1, 4 and 8 taps converge after about 220 iterations, the 12-tap LMS filter after around 300 iterations. All realisations apart from the 1-tap filter show a steady-state MSE level of about  $-70$  dBm. The 1-tap filter settles at about  $-55$  dBm potentially indicating a poor mitigation performance. Considering Figure 4.14b, the presence of the settling time is also observable as high variations of the coefficients appear at the beginning. Similarly, the number of iterations needed for convergence also yields about 220, while the coefficient of the 4-tap filter shows clutter around small values. The focus of Figure 4.14b is pointed at the lower order filter coefficient located around small values. Although the coefficients of the 8- and 12-tap filter do not fit in the illustrated range, the convergence of both filter realisations is assumed, according to Figure 4.14a.

From now on, the focus of the implementation is based on the LMS configuration with  $L = 4$  as it apparently shows almost the same performance as higher filter orders and, more importantly, represents the most efficient implementation in terms of hardware utilisation as less embedded hardware multipliers are required.

In the next step, the dependence of the parameter  $MU$  is investigated while setting  $SF$  to 16, as illustrated in Figure 4.14c and Figure 4.14d. The higher  $MU$  is chosen to be (i.e. the smaller the coefficient update output term  $\mu e[n]\mathbf{r}[n]$  will be scaled) the smoother the convergence characteristic of the coefficients will look but the more iterations it takes the LMS filter to converge the steady-state. Nevertheless, the residual error will also increase from a certain value of  $MU$ , which is according to Figure 4.14b from  $MU = 13$ , results in a worse mitigation performance. Hence,  $MU$  must be kept as small as possible.

The influence of the parameter  $SF$  is analysed by setting  $MU = 11$ , as shown in Figure 4.14e and Figure 4.14f. It is apparent from both figures that increasing the factor  $SF$  from the optimised value 16 (i.e. scaling down the output of the FIR filter) yields a longer convergence time. Increasing  $SF$  even further results ultimately in a higher residual error, which corresponds to a worse mitigation performance. The coefficient behaviour convergence on the right-hand side even reveals that with  $SF$  at 20 or 22 the first real coefficient does not converge within the 1000 iterations at all. Therefore, it can be concluded that the value of  $SF$  must be kept low in order to provide a low MSE level under a desirable convergence behaviour. Figure 4.14h

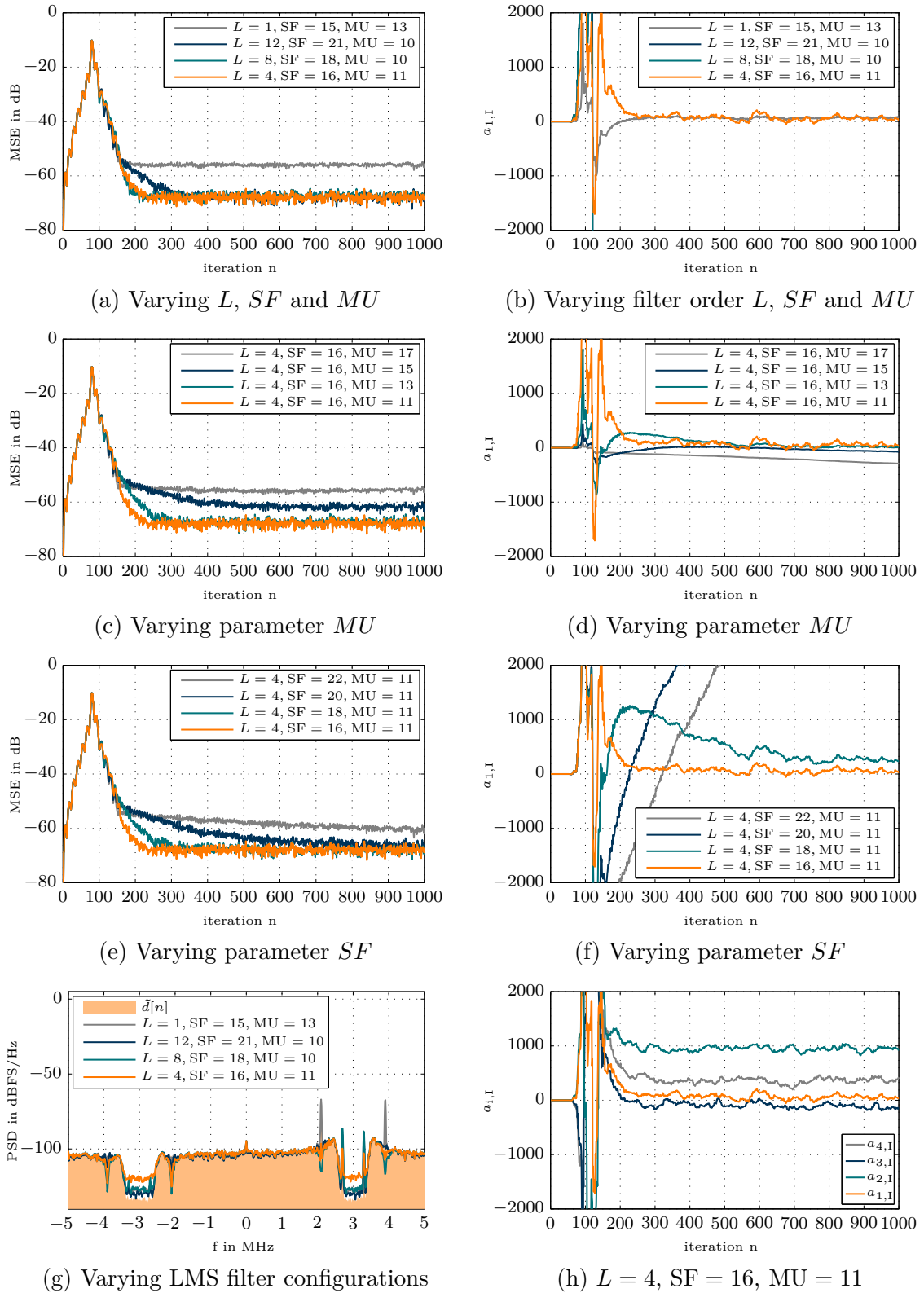


Figure 4.14: LMS filter convergence behaviour considering in (a,c,e) the MSE function  $\mathbb{E}\{|\tilde{e}[n]|^2\}$  (averaged over 15 realisations), in (b,d,f) the first real coefficient convergence behaviour, (h) all real coefficients and in (g) the PSD of  $\tilde{e}$  ( $NFFT = 1024$ , averaged over 15 realisations, Hann window)

presents the convergence behaviour of three residual real coefficients considering a LMS configuration of  $L = 4$ ,  $SF = 16$ ,  $MU = 11$ . All coefficients settle at about the same number of iterations and exhibit a similar clutter as the first coefficient.

### Constraints for LMS Filter Parameters

Optimising the LMS parameters, considering the MSE and coefficient behaviour functions only, might not necessarily lead to a good mitigation performance. Instead, the PSDs of the LMS output  $\tilde{e}[n]$  needs to be analysed in order to evaluate the ultimate mitigation of the IMD products in the frequency domain, as depicted in Figure 4.14h. The light-orange faced area illustrates the signal  $\tilde{d}[n]$  of the desired branch containing the IMD products located at 2.1 MHz and 3.9 MHz to be mitigated. The one-tap LMS filter does not show any mitigation at all, as already observed in Figure 4.14a. The LMS configuration with  $L = 4$  successfully suppresses the IMD peaks. However, notches at the locations of the mitigated IMD products are generated with the 4- and 8-tap filter, whereby the notch of  $L = 8$  is even more significant. In contrast, no notches seem to be present with higher order filters such as with  $L = 12$ . Hence, the choice is made in favour of the 4-tap LMS filter when comparing the trade-off between the mitigation capability and hardware resource utilisation. The small notch is therefore considered to be tolerable. Moreover, more detailed elaborations about the achieved mitigation is discussed based on the measurement results in Chapter 5.

It is important to note that the findings of the optimal parameters  $SF$  and  $MU$  only apply to a specific scenario. In practice, both parameters should be adjusted in order to provide a good mitigation performance within arbitrary scenarios. Concluding the subsequent findings for  $SF$  and  $MU$  of the specific scenario, general constraints can be deduced, which help to determine appropriate values, as summarised in the following:

1.  $SF + MU \geq 26$ , otherwise an overflow might occur.
2. Both  $SF$  and  $MU$  must be kept as low as possible.
3.  $SF$  is in general chosen to be higher than  $MU$ .
4. A higher LMS filter order  $L$  requires a value for  $SF$ .

## Migration to HDL Code

Referring to the implementation design flow, the following step is to migrate the NON-LIM implementation in HDL code and validate it with the aid of the *SigIO* interface. Since the implementation of the experimental MATLAB script is already established in a fixed-point arithmetic, the migration is straightforward. However, the challenge of the HDL implementation lies in the timing of the individual processing steps so that no timing constraints of the design are violated. This is important since each output sample and the filter coefficients update need to be performed within 8 cycles in regards to a chosen decimation rate of 8. Hence, one of the key points is represented by the FIR filter part performing  $2L$  parallel multiplications. The instantiation within the HDL design is outlined in Listing 4.3.

Listing 4.3: Instantiated hardware multiplier forming an FIR filter

```

1  localparam L = 4;
2  generate
3  genvar i;
4  for(i=0; i<L; i=i+1) begin: fir_filt_mult
5      MULT18X18 filter_i
6          (.P(p_i[i]), .A(r_i[i]), .B(a_i[i]));
7      MULT18X18 filter_q
8          (.P(p_q[i]), .A(r_q[i]), .B(a_q[i]));
9
10     always @(posedge clock) begin
11         p_i_reg[i] <= p_i[i];
12         p_q_reg[i] <= p_q[i];
13     end
14
15     sign_extend #(.bits_in(36), .bits_out(36+L-1)) sign_extend_p_i_reg (.in(
16         p_i_reg[i]), .out(se_p_i_reg[i]));
17     sign_extend #(.bits_in(36), .bits_out(36+L-1)) sign_extend_p_q_reg (.in(
18         p_q_reg[i]), .out(se_p_q_reg[i]));
19 end //end for-loop
20 endgenerate
21
22 always @(posedge clock) begin
23     y_i_reg[1] <= se_p_i_reg[0] + se_p_i_reg[1];
24     y_i_reg[0] <= y_i_reg[1] + se_p_i_reg[2] + se_p_i_reg[3];
25     y_q_reg[1] <= se_p_q_reg[0] + se_p_q_reg[1];
26     y_q_reg[0] <= y_q_reg[1] + se_p_q_reg[2] + se_p_q_reg[3];
27 end

```

Line 2-19 constitutes a *generate*-block, which allows multiple instances with the same name. This enables a flexible instantiation of the multipliers within the HDL code using only one Verilog instantiation statement. The simulator or the synthesis tool then elaborates the generate-block and unrolls the for-loop of line 4. The genvar variable  $i$

is an integer variable being only defined during the elaboration of the generate blocks [48, 49]. Hence, line 5-7 create  $4 \cdot 2 = 8$  multiplier primitives *MULT18X18* of the Spartan-3A architecture that are eventually synthesised to an DSP48A slice during the design implementation process [61]. Therefore, the reference distortion samples  $r_{i/q}[i]$  are multiplied by the respective coefficients  $a_{i/q}[i]$ , where  $i$  counts from 0 to 3 corresponding to the actual and the past 3 samples. After this, the individual results are stored to the registers  $p_{i/q\_reg}[i]$  at the next rising clock edge. In lines 15-16, the partial products  $p_{i/q\_reg}[i]$  are sign-extended to  $36 + L - 1 = 39$  bit to avoid potential overflows and are in lines 22-25 accordingly added together. The addition is therefore divided into two cycles, relaxing the timing constraints of the design. In this way, it takes 4 clock cycles in total to multiply and accumulate the reference distortion samples  $r_{i/q}[i]$  and the coefficients  $a_{i/q}[i]$  of the actual snapshot  $n$ . Subsequently, the 39-bit wide results  $y_{i/q\_reg}[0]$  require to be bit-truncated and right shifted by  $SF$  bit positions, as shown in Listing 4.4. This is established by performing a simple bit select of the respective 16 bits from the 36-bit result. Bit number 35 thereby corresponds to the MSB, which is the sign bit, and  $[14 + SF : SF]$  selects the scaled magnitude truncating the rightmost  $MU$  bit positions.

Listing 4.4: HDL bit-truncation and arithmetic right shift by  $SF$  bit positions

```
1 assign y_i_scaled = {y_i_reg[0][35+L-1], y_i_reg[0][14+SF:SF]};
```

In the remaining 4 cycles, the filter coefficient update takes place, as illustrated on the right-hand side in Figure 4.13. The multiply operation is implemented similarly to the FIR filter part. In contrast, the partial products  $y_{i/q}[n] \cdot r_{i/q}[n]$  are processed individually, i.e. they are accumulated with their corresponding actual coefficients  $a_{i/q}[i]$  after right shifting by  $MU$  bit positions, according to (4.4).

Once the implementation has been successfully migrated to HDL, the testbench must be created providing the file I/O interface to the *SigIO* class, as shown in Listing 4.1. The *SigIO* class stores the stimulus data to the respective input file of the testbench and compiles the simulation environment by executing *fuse* using the corresponding HDL source files. Then, the behavioural simulation is started by running the simulation executable. After finishing the simulation, the *SigIO* class reads the simulation output file back and visualises the data in the same manner as established by the pure HL simulation. If the results equal those obtained by the HL simulation, it can proceed with the design's synthesis and implementation step following the implementation design flow, whereby timing information are generated after the place and route process.



In Figure 4.15a, the PSD of the LMS filter output is obtained by the back-annotated timing simulation, which incorporates the timing information of the *PAR* process. The coefficient ports of the HDL LMS filter module, which are monitored by the testbench and supplied to the *SigIO* class, are unconnected within the NONLIM design so that they are removed by XST. Therefore, no timing information is generated, whilst Figure 4.15b depicts the coefficient convergence behaviour, which is obtained from the behavioural HDL simulation.

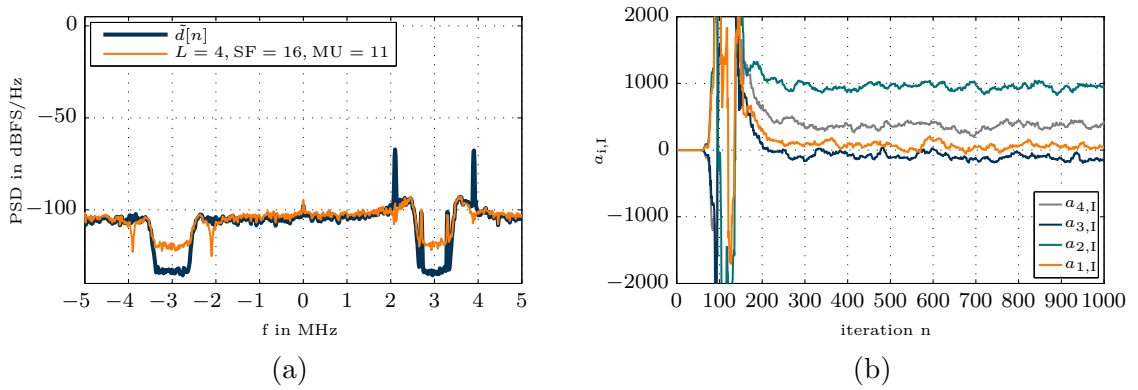


Figure 4.15: HDL simulation with  $L = 4$ ,  $SF = 16$ ,  $MU = 11$  showing (a) the PSD of  $\tilde{e}$  (timing simulation,  $NFFT = 1024$ , averaged over 15 realisations, Hann window) and (b) all real coefficients  $a_{i,I}$  (behavioural simulation)

When comparing the HDL simulation results of the PSD and the filter coefficients convergence behaviour with Figure 4.14g and Figure 4.14h, the results of both the HL and HDL timing simulation are seen to yield the same result (assuming  $L = 4$ ,  $SF = 16$ ,  $MU = 11$ ). Therefore, the FPGA programming and the practical validation on hardware can be continued.

### 4.3.7 Resource Utilisation

The amount of resources required for implementing the NONLIM algorithm strongly depends on the given requirements and the signal scenario to meet. Thereby, the FIR filter orders constitute a significant factor. The type the filter structure implementation (MAC or DA), which is used for the filter implementation, defines what kind of resources are utilised. The amount of hardware resources used for each individual implemented sub-module of NONLIM can be read from the MRP, which is generated after the *MAP* process. This is summarised in Table 4.3.

The amount of resources listed above does not incorporate fabric, which is needed for the interconnection of the sub-modules, rounding operations or pipeline stages.

Type\Module	<i>bsd_filter</i>	<i>bpr_filter</i>	<i>refmodel</i>	<i>bsr_filter</i>	<i>r/d_delay</i>	<i>lms_filter</i>
LUTs	1158	6796	160	697	0	562
FFs	1478	7294	352	883	0	878
DSP48A	22	0	6	13	0	16
Block RAM	0	0	0	0	1 (each)	0
Filter order	161	95	-	95	0	4
Architecture	MAC	DA	MAC	MAC	-	MAC

Table 4.3: Resource utilisation of the individual sub-modules of NONLIM

All filter realisations, apart from the *lms\_filter*, have a higher filter order and are symmetric. They are realised as multi-channel realisations, which utilise the DSP48A slices in a time-multiplexed, resource-efficient manner. The *lms\_filter* needs to perform the whole MAC operations within 8 cycles and utilises the DSP48A slices in a parallel way. A 4-tap LMS filter, in turn, already takes 16 DSP48A slices. This demonstrates the trade-off between efficient resource usage and a low processing latency.

Furthermore, *bpr\_filter* is the only complex-valued filter realisation and would therefore require twice as many DSP48A slices than the *bsr\_filter* by exhibiting the same filter order. For this reason, *bpr\_filter* is implemented using the DA filter architecture in order to save DSP48A slices and to achieve a sound balance between the types of utilised resources.

Table 4.4 summarises the amount of utilised resources with and without the employed NONLIM algorithm, as well as the amount of resources solely required by NONLIM. Accordingly, the amount of DSP48A slices utilised by NONLIM accounts for 57 of the total 126, which results in 38 slices still being left over after the implementation. The DA filter makes up the largest part of the occupied FPGA fabric, such as the LUTs and FFs, that is used by NONLIM. However, the saved DSP48A slices can be utilised to deploy further DSP applications.

Design Type	USRP N210 (original)	NONLIM	USRP N210 + NONLIM
LUTs	31649/47744 (66%)	9783 (20%)	40578/47744 (85%)
FFs	20007/47744 (41%)	11446 (24%)	31179/47744 (65%)
DSP48A slices	31/126 (24%)	57 (45%)	88/126 (70%)
Block RAM	41/126 (32%)	2 (2%)	43/126 (34%)

Table 4.4: Resource utilisation of NONLIM compared to total resources of the USRP N210

## 4.4 Processing Delay of NONLIM

Another important figure of merit of the implementation is the total processing delay. It is especially relevant for communications systems, such as GSM, that use a time-division multiple access (TDMA) scheme. Here, a communication channel is time-multiplexed, i.e. the time domain is divided into fixed time slots of a fixed length. If the NONLIM should be integrated, the total processing time increases and still has to meet the timing specifications of the standard.

There are different ways to determine the processing delay of a module. With regard to the IP cores, the GUI of the *CORE Generator* already calculates and displays the delay being subject to the selected core configuration. Furthermore, the induced delays of the respective HDL modules can be determined by running a timing simulation. If no timing errors are reported by the implementation tools, it can be assumed that the simulated delays are also met in practice. The individual delays induced by the sub-modules are listed in Table 4.5.

Module	delay (cycles)
<i>bsd_filter</i>	33
<i>bpr_filter</i>	16
<i>bsr_filter</i>	24
<i>lms_filter</i>	6
<i>refmodel</i>	10
<i>r_delay</i>	$49 \cdot 8$
<i>d_delay</i>	$16 \cdot 8$

Table 4.5: Delays of sub-modules and of NONLIM

According to the methodology of the NONLIM algorithm, the total processing delay is evaluated by the sum of the sub-modules in the reference branch, namely the *bpr\_filter*, the *refmodel*, and the *bsr\_filter*, as well as the *lms\_filter* module. It equates to the first 16 samples, which are dropped due to the settling of the FIR filters, plus additional delays induced by pipeline stages, rounding operations and other logic within the NONLIM module. Therefore, the total processing delay accounts for 201 cycles corresponding to  $2.01 \mu\text{s}$  with respect to the FPGA master clock of 100 MHz.

## 4.5 Complex-valued LMS Implementation

In Section 3.4, separated I/Q processing has been outlined as one of the simplifications made by the NONLIM algorithm. As it is less complex, separated I/Q processing

was assumed to be sufficient for the mitigation process. However, several components within the DCR chain introduce I/Q imbalance, which possibly leads to a worse mitigation performance of the NONLIM algorithm. For this purpose, a complex-valued LMS implementation of the LMS filter might yield the same mitigation performance, whilst using a lower filter order, and therefore would not utilise as many hardware resources. The modification that needs to be made for a complex-valued LMS implementation basically requires a complex FIR filter part, as also shown for the complex *bpr\_filter* in (4.1). That is, the complex reference distortion samples  $r_{I/Q}[i]$  additionally need to be cross-multiplied by the complex coefficients  $b_{j,I/Q}$  and subsequently subtracted or added. This yields:

$$\tilde{y}[i] = (b_{j,I}[n]r_I[i] - b_{j,Q}[n]r_Q[i]) + j(b_{j,I}[n]r_Q + b_{j,Q}[n]r_I), \quad (4.6)$$

where  $r_{I/Q}[i]$  with  $i = n, \dots, (n - L - 1)$  corresponds to the actual and the past  $L - 1$  samples and  $b_{j,I/Q}$  with  $j = 1, \dots, L$  denotes the filter coefficient set of the current snapshot  $n$ . The number of required hardware multipliers (i.e. DSP48A slices) therefore doubles for the same filter order  $L$  and additional accumulation logic is needed for adding the partial products together, respectively. The relationship between the LMS filter order  $L$  and the required DSP48A blocks is listed Table 4.6, comparing the real- and complex-valued LMS filter realisations.

$L$	real-valued	complex-valued
in general	$2L + 2L = 4L$	$4L + 2L = 6L$
1	4	6
<b>2</b>	8	<b>12</b>
3	12	18
<b>4</b>	<b>16</b>	24

Table 4.6: Utilised DSP48A slices of a real- and complex-valued LMS AF realisation

When reducing the LMS filter order from 4 to 2, the number of multipliers required for the FIR filter section would in fact stay the same. In contrast, the amount of multipliers required for the complex-valued coefficient update would reduce to in total 4 instead of 8. As a result, 4 DSP48A slices could be saved by complex LMS filter implementation.

Another important point to consider is that the additional accumulation of the partial products requires an extra clock cycle. Since the filter operation and the coefficient update already take 8 cycles, which concurrently constitutes the maximum delay, one cycle must be saved in order to accommodate the additional accumulation processing

---

cycle. When assuming a complex filter with order  $L = 2$ , the cycle, which sums up the second part of the filter output  $y_{I/Q}[n]$  in lines 23 and 25 of Listing 4.3, could potentially be used for performing the subtractions and additions of (4.6). Therefore, a complex-valued 2-tap LMS filter implementation could be realised.

## 5 Measurement Results

After having programmed the USRP N210 with the custom FPGA design, the measurements have to be carried out in order to validate the correct functionality of the NONLIM implementation. The measurements were conducted using the MATLAB-UHD class, which communicates directly with the USRP N210. Two types of measurement scenarios were investigated: A two-tone and a BPSK scenario. The experimental setup of either scenario is first introduced before the measured data is then presented.

### 5.1 Two-tone Scenario

#### Experimental Setup

In order to practically validate the functionality of the NONLIM algorithm, two-tone measurements are conducted in order to compare the results to the simulation. The experimental setup comprises two signal generators of type R&S SMY02 and of type R&S SMHU, which generate two sine signals with a power of  $-11$  dBm at the arbitrarily chosen frequencies of 92.70 MHz and 93.30 MHz, respectively. The two sine signals are added with a power combiner, which incorporates an insertion loss of approximately  $-5$  dBm. A common reference clock of 10 MHz ensures a coherent interaction between the devices.

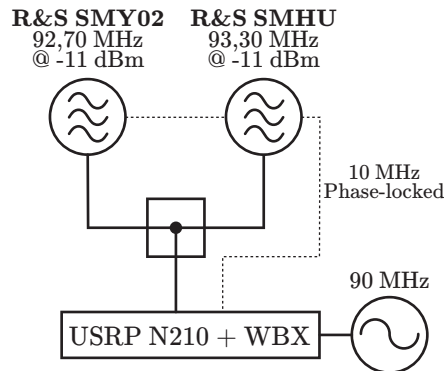


Figure 5.1: Experimental setup of two-tone scenario

The USRP N210 with the wideband WBX front-end is tuned to a centre frequency of 90 MHz and the decimation rate of 8 is chosen, resulting in a BB bandwidth of 12.5 MHz.

## Results

Figure 5.2 depicts the PSD of the measured distorted and cleansed BB signals  $\tilde{x}_{\text{dist}}$  and  $\hat{x}$ , which were obtained using the original and the modified NONLIM firmware, respectively. The PSD is computed using a 1024-point FFT with a Hann window, as well as an averaging over 15 realisations is carried out. The BB spectrum of the distorted signal  $\tilde{x}_{\text{dist}}(t)$  contains the two tones with a power of about  $-9$  dBm located at 2.7 MHz and 3.3 MHz and the resulting IMD products. These are stemming at the frequencies of 2.1 MHz and 3.9 MHz with a power of about  $-63$  dBm, which is in accordance to (2.11). Besides, the two mirror peaks at  $-2.7$  MHz and  $-3.3$  MHz of  $-54$  dBm of the mirror frequency emerge due to the I/Q imbalance, which is mainly induced by the mixer in the DDC chain. As a further RF impairment, a small DC offset peak appears at DC, which exhibits a power of  $-92$  dBm.

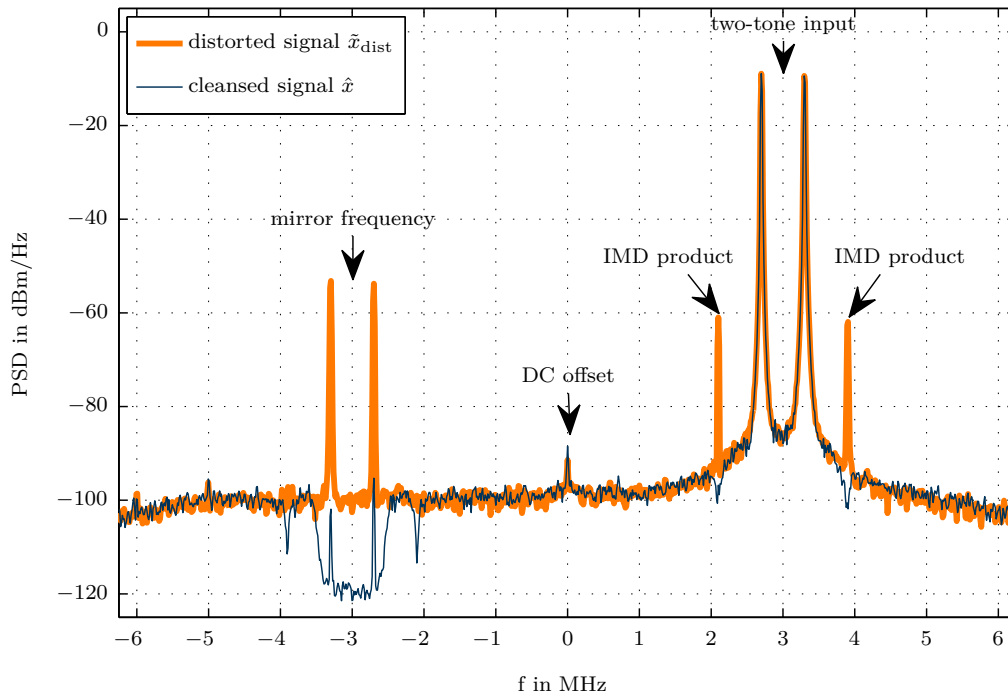


Figure 5.2: Measurement BB spectrum before and after mitigation using parameters  $L = 4$ ,  $SF = 16$ , and  $MU = 11$  ( $NFFT = 1024$ , averaged over 15 realisations, Hann window).

Regarding the cleansed BB spectrum of the signal  $\hat{x}$ , it is apparent that the NONLIM algorithm has successfully mitigated both IM3 products. However, the notches located at the positions of the suppressed IMD products are still present in practice, showing a depth of approximately  $-6$  dB. Moreover, it is interesting to observe that the notches also appear on the opposite side, which potentially presents an issue that is caused by separated I/Q processing. Furthermore, the mirror frequency is also suppressed due to the real-valued *bsd\_filter*, which shows a symmetric filter behaviour.

## 5.2 BPSK Scenario

### Experimental Setup

Until now, it has been assumed that the down-converted BB spectrum only consists of the two-tone interferer and the induced IMD products (disregarding any other concurrently existing RF impairments). In this sense, a binary phase shift keying (BPSK) scenario is invented, where the target spectrum of a weak modulated signal is adjacent to the strong two-tone interferer, as already illustrated in Figure 3.1. Hence, the IMD product that is created left to the two tones hits the target band, which needs to be mitigated by the NONLIM algorithm. BPSK is chosen as the modulation scheme of the target band and is generated with a vector signal generator of type R&S SMU200A. The BPSK signal is modulated at a symbol rate 788 kSymbols/s and the roll-off factor of the raised-cosine filter is set to 0.20. The BPSK signal is coupled into the existing RF path of the two-tone signal using a directional coupler with a coupling loss of approximately 40 dB at a frequency of 91 MHz, as illustrated in Figure 5.1. Accordingly, the centre frequency of the BPSK signal is set to 91.90 MHz and the output power to  $-13$  dBm.

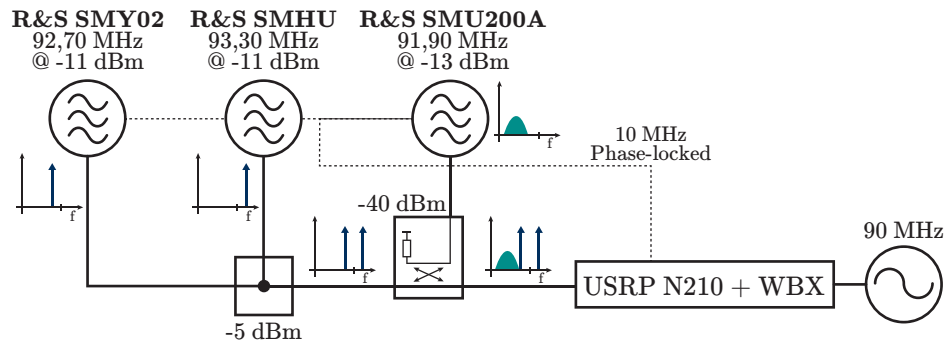


Figure 5.3: Experimental setup of BPSK scenario



## Results

The result of the measurements of the BPSK scenario is shown in Figure 5.4, comparing the spectra that were obtained by the original and the modified NONLIM firmware. In consideration of the distorted signal spectrum of  $\tilde{x}_{\text{dist}}$ , one of the IMD peaks amounts to a power of  $-62$  dBm and apparently hits the target spectrum of the BPSK signal, whose maximum signal component accounts for approximately  $-73$  dBm. The difference of the power levels therefore amounts to  $-11$  dB, which potentially makes a correct demodulation of the BPSK signal impossible. Furthermore, the notch is again present at the mitigated IMD product's position showing a depth of about  $-12$  dBm. At the same time, the spectrum of the BPSK signal seems to be slightly raised. Moreover, the IMD product on the right-hand side of the two tones is not fully mitigated to the noise floor, which results in a suppression value of approximately  $-18$  dBm. A possible explanation for this behaviour might be based on the fact that the LMS filter seems only to adapt the reference distortions to a certain level of the signal contained in the desired branch.

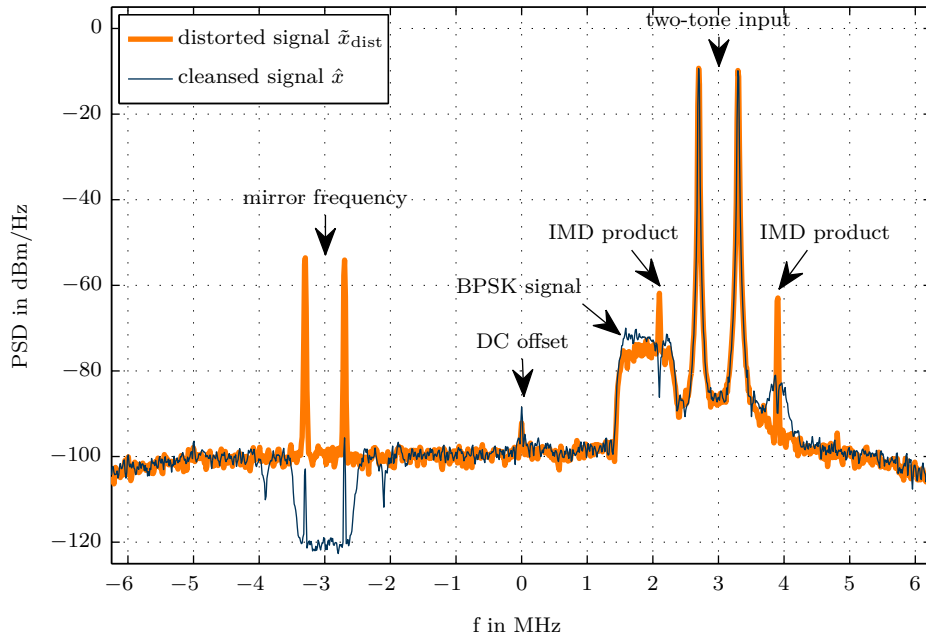


Figure 5.4: Measurement BB spectrum before and after mitigation using parameters  $L = 4$ ,  $SF = 16$ , and  $MU = 11$  ( $NFFT = 1024$ , averaged over 15 realisations, Hann window).

### 5.3 Discussion

Regarding the measurement of the two-tone scenario, the results are promising as the induced IMD products are successfully mitigated. The mitigation performance of the timing simulation of Figure 4.15a and of the practical validation of Figure 5.2 have proven to be equal. Hence, the correct function of the NONLIM module is shown and the implementation is proven to function correctly.

The notches at the positions of the mitigated IMD products are similarly present in practice, suggesting an issue that is potentially LMS-related and not implementation-specific. In order to analyse how the LMS filter adapts the reference non-linearity  $\tilde{r}[n]$  to subtract it from the desired signal  $\tilde{d}[n]$ , the PSDs of the filtered out interferer signal  $\tilde{u}[n]$  and of the generated reference non-linearity  $\tilde{r}[n]$  are shown in Figure 5.5a. Moreover, Figure 5.5b depicts the PSD of the distorted signal  $\tilde{d}[n]$ , of the cleansed LMS output  $\tilde{e}[n]$  and of the adjusted reference non-linearity  $\tilde{r}_{\text{adj}}[n]$ .

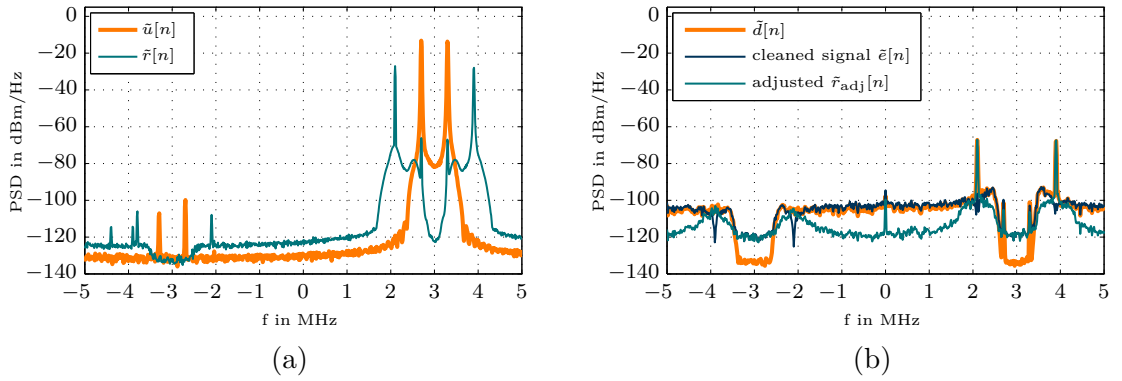


Figure 5.5: Simulation analyses of the adapted reference non-linearity showing the PSD of (a) the filtered interferer signal  $\tilde{u}[n]$  and the generated reference non-linearity  $\tilde{r}[n]$ , and (b) the distorted signal  $\tilde{d}[n]$ , adjusted reference non-linearity of  $\tilde{r}[n]$  and cleansed output  $\tilde{e}[n]$  ( $NFFT = 1024$ , averaged over 15 realisations, Hann window)

Considering the adjusted reference distortions  $\tilde{r}_{\text{adj}}[n]$  in Figure 5.5b, the two IMD products are successfully adapted in order to subtract them from the distorted signal  $\tilde{d}[n]$ . Likewise, spurious components of the unscaled reference distortions  $\tilde{r}[n]$  are equally lifted to the noise floor, which are also subtracted from  $\tilde{d}[n]$ . Hence, the notches emerging at the mitigated IMD products' positions are supposedly caused by these spurious frequency components. These are incorporated in the regenerated reference distortions  $\tilde{r}[n]$  as they seem to be jointly scaled together with the IMD products. Furthermore, the original two tones are also contained in the adjusted reference dis-

tortions  $\tilde{r}_{\text{adj}}[n]$  since they are also included in the unscaled reference distortions  $\tilde{r}[n]$ , as shown in Figure 5.5a.

To put it simply, the spurious frequency components of the  $\tilde{r}_{\text{adj}}[n]$  originate from an impure reference distortions signal  $\tilde{r}[n]$ , which contains also other frequency components than the IMD products. In turn, these do emerge from an imperfectly filtered-out interferer signal  $\tilde{u}[n]$ , which was filtered by the *bpr\_filter*. The *bpr\_filter* has a slope of finite steepness and therefore does not perfectly separate the interferer band from the residual spectrum, as shown in Figure 5.5a. After applying this interferer signal  $\tilde{u}[n]$  to the reference model, the spurious components propagate through the reference branch and are apparent in the reference distortions  $\tilde{r}[n]$ . The spectrum of these shows an even more flattened envelope below a level of approximately  $-80$  dBm instead of two sharp two-tone peaks, as illustrated in Figure 5.5a. To account for that, one can tie the filter specifications for *bpr\_filter* and choose a lower transition width between the stop- and passband in order to increase the steepness of the slopes of the filter. Therefore, a higher filter order is required and more hardware resources are utilised. The frequency components, which are raised to the noise floor on the negative side of the spectrum at  $-3.9$  MHz and  $-2.1$  MHz, might be caused due to the separated I/Q processing. A complex-valued I/Q processing, being subject to future work, will potentially account for this.

In terms of an optimised mitigation performance and convergence behaviour, the LMS parameters  $L = 4, SF = 16, MU = 11$  are strictly optimised towards the two-tone scenario. However, the spurious IMD product, which hits the target spectrum of the BPSK scenario and possibly disrupts the demodulation of the BPSK signal, is successfully mitigated. In consideration of the practical application of NONLIM, this demonstrates that a good mitigation performance can be achieved even with non-optimised LMS parameters  $SF$  and  $MU$ , which are commonly not known in practice. However, a notch remains at the position of the removed IMD peak in the BPSK spectrum, which might undo the improvement of the removed IMD product. This needs to be investigated within the scope of BER measurements, which compare the BERs of the demodulated BPSK signal: On the one hand by means of including the IMD product on top of the BPSK spectrum and on the other hand by excluding the IMD product but with the notch.

## 6 Future Work

### 6.1 Complex-valued LMS Filter

The implementation of a complex-valued LMS filter can exemplary be established by modifying the real-valued FIR filter part of the LMS filter so it is complex-valued. This might provide an advantage as less hardware resources are needed as a smaller filter order  $L$  is required. Accordingly, the influence of the NONLIM algorithm with a complex-valued LMS filter should be investigated with respect to the mitigation behaviour in order to determine the equivalent complex-valued LMS filter order  $L$ . Thus, the occurrence of various mitigation effects needs to be studied to see whether

1. the notches at the positions on both sides of the spectrum are still present,
2. the IMD peak on the right-hand side of the two tones can also be mitigated with regards to the BPSK scenario,
3. the BPSK spectrum is still slightly being raised.

Appropriate BER measurements will support the evaluation of the performance of the complex-valued LMS implementation, which is subject to future work.

### 6.2 Extensions to Arbitrary Interferer Signals

In prior work, it has always been assumed a basic scenario, where a strong, blocking two-tone interferer induces two distinct IMD products being adjacent to the interferer. In practice, a two-tone interferer does not constitute a realistic signal since it does not bear any information. Instead, the non-linear out-of-band distortions of modulated interfering signals become noticeable in different ways.

With regard to the GMSK modulation that is used in GSM, a single GMSK-modulated signal does not cause any non-linear distortions. This is due to its property of showing a constant *complex envelope* [62], which corresponds to a theoretical PAPR of 0 dB [63]. Therefore, the PAPR is an important metric for the envelope fluctuations

of a modulated signal. Accordingly, it gives an indication of the non-linear distortions that are induced by non-linear components [63]. However, if two GMSK-modulated signals are positioned next to each other, the PAPR changes to 3.01 dB, which likewise corresponds to the PAPR of a two-tone signal. This indicates a similar nature of the appearance of non-linear distortions between both. The non-linear distortions that are induced by two neighbouring GSM channels and the mitigation of those has been investigated by [7]. Thereby, the BER performance was improved by applying the NONLIM algorithm to a real measured signal, which contains two GSM channels and its induced IMDs, by means of offline MATLAB processing.

In order to demonstrate the influence of the PAPR with respect to non-linear distortions, two further measurements have been carried out with the USRP N210. The first comprises two adjacent GMSK-modulated signals, which should represent two neighbouring GSM channels, and the second a BPSK-modulated interferer. The results are depicted in Figure 6.1, showing the induced distortions of the two modulated GMSK as well as the BPSK signal.

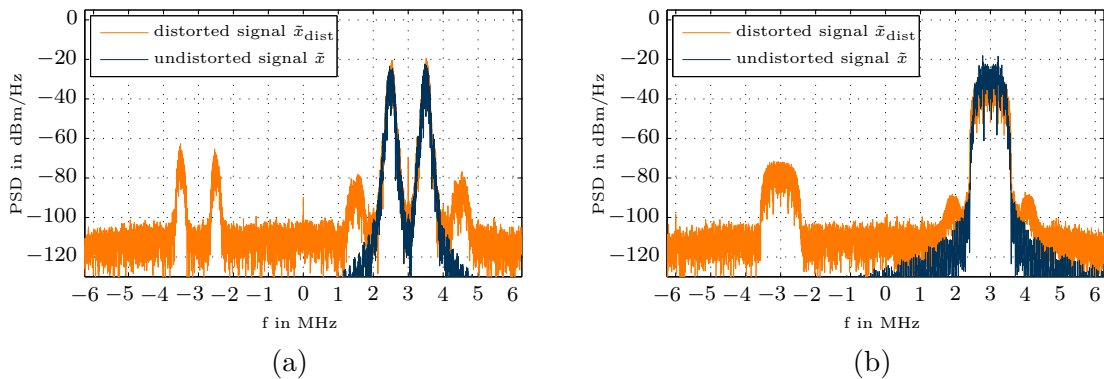


Figure 6.1: Measurement of BB spectrum showing the non-linear distortions of (a) two adjacent GMSK-modulated and (b) a BPSK-modulated interferer ( $NFFT = 8192$ , Hann window)

The neighbouring GMSK-modulated GSM channels represent a two-tone-like signal scenario and are positioned at 2.5 MHz and 3.5 MHz in the BB. Each of them is modulated with a symbol rate of 271 kSymbols/s with a peak power of  $-25$  dBm. It can be observed that the two distinct IMD products (in the form of GMSK spectrum) appear at the frequencies 1.5 MHz and 4.5 MHz. This is in accordance with the two-tone case of (2.11).

In contrast, the non-linear distortions, which are induced by the BPSK-modulated interferer in Figure 6.1b, appear as spectral regrowth that is closely located to the BPSK-modulated signal. The signal is modulated with a symbol rate of 787 kSymbols/s

and the roll-off factor of the raised-cosine filter is set to 0.50. The PAPR thereby accounts for 4.1 dB.

In terms of the mitigation by the NONLIM algorithm, all FIR filters have to be adjusted to the frequency range of the modulated signals, which are considered as the interferer. In this way, they can be isolated properly from the residual signal spectrum and the reference distortions can artificially be regenerated by the *refmodel* block. Once this modification is made, the NONLIM algorithm should run as expected, assuming that the LMS filter parameters  $L$ ,  $SF$  and  $MU$  are properly adapted. Nevertheless, this signal scenario calls for further investigation and will be subject to future work.

### 6.3 Runtime Reconfigurability

The LMS parameters  $SF$  and  $MU$  were identified as significant to the mitigation performance and the convergence behaviour of NONLIM. This was elaborated by the simulations during the implementation process of the LMS filter module. Since both parameters strictly depend on the overall signal scenario at hand, it is not desired to synthesise the design with fixed values for  $SF$  and  $MU$ , but rather to be able to adjust both parameters during runtime. Therefore, this way of reconfiguring the NONLIM algorithm is referred to as runtime reconfigurability. Additionally, the option of turning off both the addition of the interferer signal, which was filtered out by the *bpr\_filter*, and the bypassing of the entire NONLIM block within the DDC chain should be given. Furthermore, reconfiguring the stop- and passbands during runtime would be superior in order to make NONLIM applicable to arbitrary interferer signals.

Changing the filter characteristic requires to redesign the filter coefficients. A recomputation of those on the FPGA would not be a feasible option due to the complexity of the filter design algorithms. Instead, the filter design could be established on the host PC. A technology called partial reconfiguration allows for modifying regions of an operating FPGA design by loading a partial configuration BIT file onto the FPGA device during runtime [64]. In this way, the filter coefficients could be transferred from the PC to the FPGA device. Thereafter, the coefficients are stored in a block RAM, where they are read from by the respective filter module. The online coefficient reload capability of the filter module is configured by the *CORE Generator* GUI of the *FIR Compiler* during the generation process of the IP core. This allows for modifying the filter coefficient sets to adapt the filter characteristics during runtime.

For the reconfiguration of the remaining parameters, a control interface is required to transmit the settings of the NONLIM block from the host PC. Fortunately, a common

control interface is already employed in the architecture of the USRP N210, which is denoted as *settings bus interface*. The *settings bus interface* is implemented as a write-only address/data bus being part of the native Wishbone (WB) bus protocol [65]. The WB bus serves as a communication interface between the FPGA and the ZPU softcore processor, which controls the data flow of the USRP. The *settings bus interface* comprises an 8-bit address, a 32-bit data word and a strobe signal, which allows for a transmission of 32-bit data words from the host PC to the FPGA via the UHD interface. These three signals are distributed over each sub-module of the DDC and the direct up-conversion (DUC) chain in the FPGA firmware, as well as to the `custom_nonlim_rx.v` module. Hence, all relevant NONLIM settings parameters such as the LMS parameters  $SF$  and  $MU$ , as well as the flags *flag\_add\_signal* and *flag\_deactivate\_nonlim* are stored in registers that are controlled by the *settings bus interface*. For example, Listing 6.1 demonstrates how the NONLIM parameter registers are being set using the *settings bus interface*.

Listing 6.1: Setting NONLIM parameter registers using the settings bus interface

```

1  always @(posedge clock) begin
2      if(set_stb) begin
3          case(set_addr)
4              1: flag_activate_nonlim <= set_data[0];
5              2: flag_add_signal <= set_data[0];
6              3: sf <= set_data[5:0];
7              4: mu <= set_data[5:0];
8          endcase
9      end
10 end
11
12 assign bb_sample = flag_activate_nonlim ? bb_sample_reg : ddc_out_sample;
```

With a strobe signal `set_stb` coming in at a rising clock edge, a case statement checks the address `set_addr` of the incoming setting and assigns the value of `set_data` to the corresponding NONLIM parameter register. As an example, line 12 demonstrates how the NONLIM module is activated or deactivated depending on the value that was being set in the register *flag\_activate\_nonlim*. If *flag\_activate\_nonlim* is set to 1, the modified sample *bb\_sample\_reg* will be assigned to the NONLIM output *bb\_sample*, otherwise the original sample *ddc\_out\_sample* coming from the DDC chain is simply bypassed.

## 7 Conclusion

The focus of this thesis is on non-linear distortions in wide-band receivers and the mitigation algorithm that is implemented on an FPGA, which resides on the SDR-platform USRP N210. The feed-forward algorithm in the digital domain is applied to clean the baseband signal of distortions. In terms of the software-defined radio application, the host PC is not involved in the mitigation process and is therefore offloaded.

First, the emergence of the intermodulation distortion products in the analogue front-end is explained by means of a two-tone example. The NONLIM algorithm artificially regenerates the reference distortions in the digital domain using a polynomial, memoryless model. An LMS AF adjusts the reference distortions in their magnitude and phase (assuming a complex-valued LMS filter) and subtracts them from the distorted signal. The NONLIM hardware implementation is a pure-digital mitigation approach, which has not been discussed in the literature so far.

It has been outlined that the architecture of FPGA benefits from an inherent flexibility that allows for a rapid implementation and evaluation of novel signal processing algorithms on hardware. After an analysis of the existing architecture of the USRP N210, an implementation design flow has been deduced to integrate the mitigation module into the direct down-conversion chain of the existing FPGA firmware. A top-bottom approach has been followed starting from a HL MATLAB implementation and ending up with a low-level Verilog HDL implementation. Before proceeding to the next lower implementation stage within the implementation design flow, the results have been verified by a MATLAB-HDL class that interfaces to the HDL testbenches, which are used for the validation of the implemented HDL modules. Thereby, practical measurement data has been used to carry out both HL and HDL simulations. Eventually, after programming the FPGA device, the correct functionality of the implemented NONLIM algorithm has been successfully validated in practice for both a two-tone and a BPSK scenario.

In conclusion, the implementation of the mitigation algorithm is a sophisticated mitigation technique for cleaning a down-converted BB spectrum of non-linear distortions



---

in real-time. Therefore, the effective linearity of the RF front-end is increased. This may lead to a significant improvement in the BER performance of cleansed modulated signals, as well as to an enhanced sensing reliability in the context of CR.

The implementation of the NONLIM algorithm on the USRP N210 and the results have been published in [66].

# Acronyms

<b>1C</b>	one's complement
<b>2C</b>	two's complement
<b>ADC</b>	analogue-to-digital converter
<b>AF</b>	adaptive filter
<b>AM/AM</b>	amplitude-to-amplitude
<b>AM/PM</b>	amplitude-to-phase
<b>ASIC</b>	application-specific integrated circuit
<b>BB</b>	baseband
<b>BB-A</b>	baseband amplifier
<b>BER</b>	bit error rate
<b>BIN</b>	binary
<b>BIT</b>	bitstream
<b>BPR</b>	bandpass-reference
<b>BPSK</b>	binary phase shift keying
<b>BSD</b>	bandstop-desired
<b>BSR</b>	bandstop-reference
<b>BW</b>	bandwidth
<b>CAE</b>	computer-aided engineering
<b>CE</b>	clock enable
<b>CIC</b>	cascaded integrator-comb
<b>CLB</b>	configurable logic block
<b>CORDIC</b>	COordinate Rotation DIgital Computer
<b>CR</b>	cognitive radio
<b>DA</b>	Distributed Arithmetic
<b>DAC</b>	digital-to-analogue converter

---

<b>DCM</b>	digital clock manager
<b>DCR</b>	direct-conversion receiver
<b>DDC</b>	direct down-conversion
<b>DSP</b>	digital signal processing
<b>DUC</b>	direct up-conversion
<b>EDA</b>	electronic design automation
<b>FDM</b>	frequency-division multiplexing
<b>FDMA</b>	frequency-division multiple access
<b>FF</b>	flip-flop
<b>FFT</b>	fast Fourier transform
<b>FIFO</b>	first in, first out
<b>FIR</b>	finite impulse response
<b>FPGA</b>	field programmable gate array
<b>FSM</b>	finite state machine
<b>GigE</b>	gigabit ethernet
<b>GMSK</b>	Gaussian minimum-shift keying
<b>GPP</b>	general-purpose processor
<b>GPS</b>	Global Positioning System
<b>GSM</b>	Global System for Mobile Communications
<b>GTL</b>	gunning transceiver logic
<b>GUI</b>	graphical user interface
<b>HBF</b>	halfband filter
<b>HDL</b>	hardware description language
<b>HL</b>	high-level
<b>I/O</b>	input/output
<b>I/Q</b>	in-phase/quadrature
<b>IC</b>	integrated circuit
<b>IF</b>	intermediate frequency
<b>IIP2</b>	input-referred second-order intercept point
<b>IIP3</b>	input-referred third-order intercept point
<b>IM3</b>	third-order intermodulation

---

<b>IMD</b>	intermodulation distortion
<b>IOB</b>	input/output block
<b>IP</b>	Intellectual Property
<b>IP2</b>	second-order intercept point
<b>IP3</b>	third-order intercept point
<b>LMS</b>	least mean square
<b>LNA</b>	low-noise amplifier
<b>LO</b>	local oscillator
<b>LS</b>	least squares
<b>LSB</b>	least significant bit
<b>LTI</b>	linear time-invariant
<b>LUT</b>	lookup table
<b>LVC MOS</b>	low voltage complementary metal oxide semiconductor
<b>LVDS</b>	low-voltage differential signalling
<b>MAC</b>	Multiply-Accumulate
<b>MMSE</b>	minimum mean square error
<b>MRP</b>	map report
<b>MSB</b>	most significant bit
<b>MSE</b>	mean squared error
<b>NCD</b>	native circuit description
<b>NCO</b>	numerically controlled oscillator
<b>NGD</b>	native generic database
<b>NONLIM</b>	non-linearly induced interference mitigation
<b>PAPR</b>	peak-to-average power ratio
<b>PC</b>	personal computer
<b>PCB</b>	printed circuit board
<b>PCI</b>	peripheral component interface
<b>PSD</b>	power spectral density
<b>PU</b>	primary user
<b>RAM</b>	random-access memory

---

<b>RF</b>	radio frequency
<b>RFID</b>	radio-frequency identification
<b>ROM</b>	read-only memory
<b>RTL</b>	register transfer level
<b>SDF</b>	standard delay format
<b>SDR</b>	software-defined radio
<b>SFDR</b>	spurious-free dynamic range
<b>SNR</b>	signal-to-noise ratio
<b>SRL</b>	shift register logic
<b>SU</b>	secondary user
<b>TDM</b>	time division multiplex
<b>TDMA</b>	time-division multiple access
<b>UCF</b>	user constraints file
<b>UHD</b>	Universal Hardware Driver
<b>UMTS</b>	Universal Mobile Telecommunications System
<b>USB</b>	Universal Serial Bus
<b>USRP</b>	Universal Software Radio Peripheral
<b>UUT</b>	unit under test
<b>WB</b>	Wishbone
<b>WSS</b>	wide-sense stationary
<b>XST</b>	Xilinx Synthesis Tool

# Appendix

## A Experimental Setup

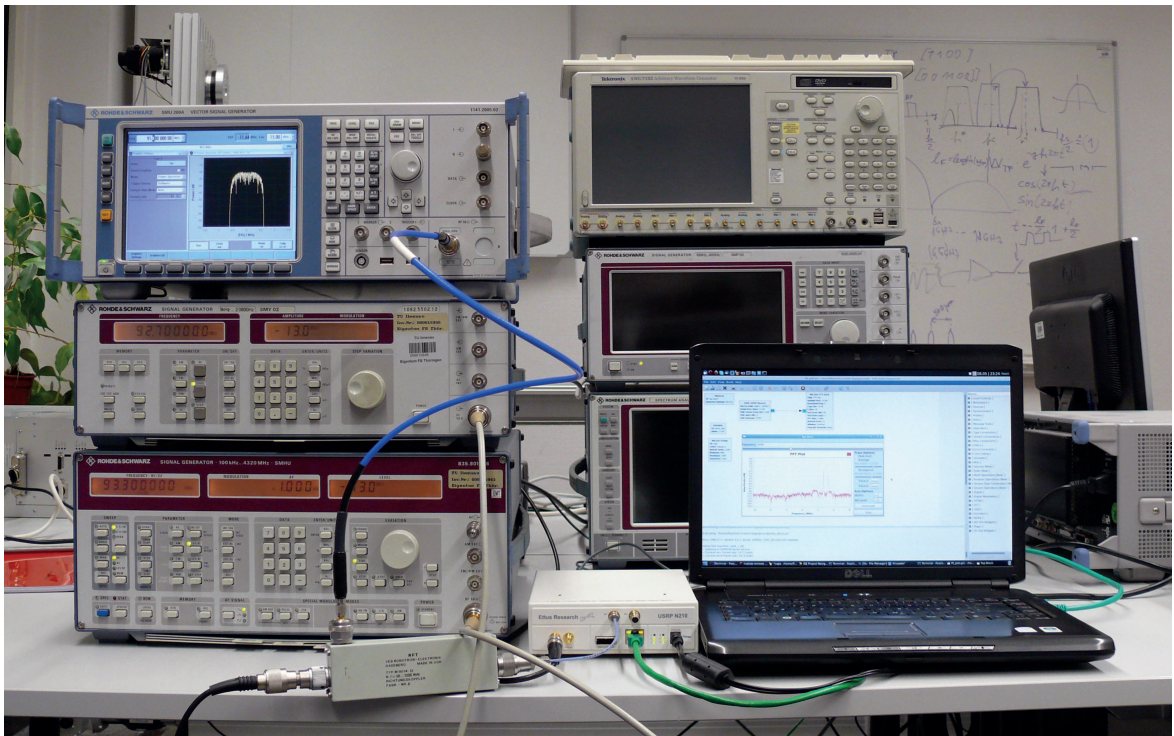


Figure A.1: Photograph of the experimental setup of the BPSK scenario

## Bibliography

- [1] Timothy B. Lee, “How Software-defined Radio could Revolutionize Wireless,” Ars technica. [Online]. Available: <http://arstechnica.com/tech-policy/2012/07/how-software-defined-radio-could-revolutionize-wireless/>
- [2] P. Kenington, *RF and Baseband Techniques for Software Defined Radio*, ser. Artech House mobile communications series. Artech House, 2005.
- [3] M. Grimm, R. K. Sharma, M. a. Hein, and R. S. Thomä, “DSP-based Mitigation of RF Front-end Non-linearity in Cognitive Wideband Receivers,” *Frequenz*, vol. 6, no. 9-10, pp. 303–310, Jan. 2012.
- [4] M. Valkama, A. Shahed Hagh Ghadam, L. Anttila, and M. Renfors, “Advanced Digital Signal Processing Techniques for Compensation of Nonlinear Distortion in Wideband Multicarrier Radio Receivers,” *IEEE Transactions on Microwave Theory and Techniques*, vol. 54, no. 6, pp. 2356 –2366, June 2006.
- [5] M. Valkama, A. Springer, and G. Hueber, “Digital Signal Processing for Reducing the Effects of RF Imperfections in Radio Devices - An overview,” *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pp. 813–816, May 2010.
- [6] P. McGuiggan, *GPRS in Practice: A Companion to the Specifications*. Wiley, 2005.
- [7] M. Grimm, R. K. Sharma, M. Hein, R. S. Thomä, and R. Zemmari, “Improved BER Performance in GSM by Mitigating Non-linear Distortions in the Receiver,” in *Proceedings of European Microwave Conference (EuMC)*, Oct. 2013, accepted for presentation at EuMW2013.
- [8] H. Venkataraman and G. Muntean, *Cognitive Radio and Its Application for Next Generation Cellular and Wireless Networks*, ser. Lecture notes in electrical engineering. Springer Netherlands, 2012.



- 
- [9] E. Keehr and A. Hajimiri, "Equalization of Third-Order Intermodulation Products in Wideband Direct Conversion Receivers," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 12, pp. 2853–2867, Dec. 2008.
  - [10] U. Meyer-Baese, *Digital Signal Processing with Field Programmable Gate Arrays*, ser. Signals and Communication Technology. Springer, 2007.
  - [11] H. Oba, M. Kim, and H. Arai, "FPGA Implementation of LMS and N-LMS Processor for Adaptive Array Applications," in *International Symposium on Intelligent Signal Processing and Communications ISPACS '06*, Dec. 2006, pp. 485–488.
  - [12] *Universal Software Radio Peripheral*, Ettus Research LLC. [Online]. Available: <http://www.ettus.com/>
  - [13] S. Mirabbasi and K. Martin, "Classical and Modern Receiver Architectures," *Communications Magazine, IEEE*, vol. 38, no. 11, pp. 132–139, 2000.
  - [14] Ettus Reseach, *Datasheet USRP N200/N210 Networked Series*. [Online]. Available: [https://www.ettus.com/content/files/07495\\_Ettus\\_N200-210\\_DS\\_Flyer\\_HR\\_1.pdf](https://www.ettus.com/content/files/07495_Ettus_N200-210_DS_Flyer_HR_1.pdf)
  - [15] J. Barry, E. Lee, and D. Messerschmitt, *Digital Communication*. Kluwer Academic Pub, 2004.
  - [16] A. Abidi, "Direct-conversion Radio Transceivers for Digital Communications," in *Solid-State Circuits Conference, 1995. Digest of Technical Papers. 41st ISSCC, 1995 IEEE International*, 1995, pp. 186–187, 363–4.
  - [17] J. Pedro and N. Carvalho, *Intermodulation Distortion in Microwave and Wireless Circuits*, ser. Artech House Microwave Library. Artech House, 2003.
  - [18] R. A. Losada and I. The Mathworks, "Digital Filters with MATLAB," 2008.
  - [19] S. Haykin and B. Van Veen, *Signals and Systems, 2005 Interactive Solutions Edition*. Wiley, 2005.
  - [20] *MATLAB Documentation Center*, The MathWorks, Inc. [Online]. Available: <http://www.mathworks.com/help/documentation-center.html>
  - [21] *Programs for Digital Signal Processing*, IEEE press, Algorithm 5.1, 1979.
  - [22] S. Haykin, *Adaptive Filter Theory*, ser. Prentice-Hall information and system sciences series. Prentice Hall, 2002.

- 
- [23] L. Tan and J. Jiang, *Digital Signal Processing: Fundamentals and Applications*. Elsevier Science, 2013.
- [24] Xilinx, Inc., “Xilinx DS534, FIR Compiler v5.0, Data Sheet,” 2011.
- [25] R. Dubey, *Introduction to Embedded System Design Using Field Programmable Gate Arrays*. Springer London, Limited, 2009.
- [26] A. Biere, D. Kroening, G. Weissenbacher, and C. Wintersteiger, *Digitaltechnik - Eine praxisnahe Einführung*, ser. Springer-Lehrbuch. Springer-Verlag Berlin Heidelberg, 2008.
- [27] M. Wannemacher, *Das FPGA-Kochbuch*. Internat. Thomson Publ., 1998.
- [28] Xilinx, Inc., “Data Sheet Xilinx DS610 Spartan-3A DSP FPGA Family,” pp. 1–101, 2010.
- [29] Xilinx, Inc., “Spartan-3 Generation FPGA User Guide (UG331),” vol. 331, 2011.
- [30] Xilinx, Inc., “Xilinx ISE In-Depth Tutorial (UG695),” 2012.
- [31] Xilinx, Inc., “Xilinx Development System Reference Guide,” 2008.
- [32] Xilinx, Inc., “Xilinx XST User Guide for Virtex-4, Virtex-5, Spartan-3, and Newer CPLD Devices,” 2008.
- [33] Xilinx, Inc., “Xilinx Synthesis and Simulation Design Guide (UG626), Designing Field Programmable Gate Arrays (FPGA devices) with Hardware Description Languages (HDLs).” 2012.
- [34] S. Palnitkar, *Verilog Hdl: A Guide to Digital Design and Synthesis*. SunSoft Press, 2003.
- [35] Xilinx, Inc., “IP Release Notes Guide List of IP Cores (XPT025),” vol. 025, pp. 1–47, 2012.
- [36] S. Bailey, “Comparison of VHDL, Verilog and SystemVerilog,” 2003.
- [37] Ettus Research, *UHD wiki*. [Online]. Available: <http://code.ettus.com/redmine/ettus/projects/uhd/wiki>
- [38] Xilinx, Inc., “Xilinx UG332 Spartan-3 Generation Configuration User Guide,” vol. 332, 2009.

- 
- [39] Xilinx, Inc., “Xilinx UG431 XtremeDSP DSP48A for Spartan-3A DSP FPGAs, User Guide,” vol. 431, pp. 1–52, 2008.
- [40] Texas Instruments, *Datasheet DUAL CHANNEL 14-BITS 125/105/80/65 MSPS ADC WITH DDR LVDS/CMOS OUTPUTS (Rev. C)*, 2012.
- [41] R. Andraka, “A Survey of CORDIC Algorithms for FPGA Based Computers,” 1998, pp. 191–200.
- [42] M. P. Donadio, “CIC Filter Introduction,” vol. 1, no. July, pp. 1–6, 2000.
- [43] L. Doolittle, “Filtering and Decimation by Eight in an FPGA for SDR and Other Applications,” no. Cic, pp. 1–8, 2006.
- [44] Mailinglist usrp-users, “Discussion about Purpose of 24-bit CORDIC in DDC chain.” [Online]. Available: <http://thread.gmane.org/gmane.comp.hardware.usrp.e100/5785/>
- [45] M. Frerking, *Digital Signal Processing In Communications Systems*. Springer, 1994.
- [46] R. Lyons, *Understanding Digital Signal Processing*. Pearson Education, 2010.
- [47] R. Woods, J. McAllister, Y. Yi, and G. Lightbody, *FPGA-based Implementation of Signal Processing Systems*. Wiley, 2008.
- [48] Design Automation Standards Committee/IEEE Computer Society, “IEEE Std 1364-2001, IEEE Standard Verilog Hardware Description Language,” pp. 1–879, 2003.
- [49] S. Sutherland, *Verilog-2001 Quick Reference Guide*. Sutherland HDL, Inc., 2001.
- [50] *MATLAB Documentation Center - Fixed-Point Designer*, The MathWorks, Inc. [Online]. Available: <http://www.mathworks.com/help/fixedpoint/index.html>
- [51] H. Flügel, *FPGA-Design mit Verilog*. Oldenbourg Wissensch.Vlg, 2010.
- [52] Xilinx, Inc., “ISim User Guide,” vol. 660, 2012.
- [53] *MATLAB Documentation Center - firpm function*, The MathWorks, Inc. [Online]. Available: <http://www.mathworks.com/help/signal/ref/firpm.html>

- 
- [54] Ettus Research, *UHD - USRP2 and N2X0 Series Application Notes*. [Online]. Available: [http://files.ettus.com/uhd\\_docs/manual/html/usrp2.html](http://files.ettus.com/uhd_docs/manual/html/usrp2.html)
- [55] Ettus Research, *UHD - Calibration Application Notes*. [Online]. Available: [http://files.ettus.com/uhd\\_docs/manual/html/calibration.html](http://files.ettus.com/uhd_docs/manual/html/calibration.html)
- [56] Xilinx, Inc., “Xilinx User Guide XtremeDSP for Virtex-4 FPGAs (UG073),” vol. 431, pp. 1–124, 2008.
- [57] Xilinx, Inc., “The Role of Distributed Arithmetic in FPGA-based Signal Processing,” 1996.
- [58] Xilinx, Inc., “Xilinx Timing Closure User Guide (UG612),” 2011.
- [59] Xilinx, Inc., “Xilinx LogiCORE IP FIFO Generator v9.2 (PG057),” pp. 1–307, 2012.
- [60] A. Oppenheim and R. Schaffer, *Discrete-time Signal Processing*, ser. Prentice-Hall signal processing series. Prentice Hall, 2010.
- [61] Xilinx, Inc., “Using Embedded Multipliers in Spartan-3 FPGAs v1.1 5/03 application note (XAPP467),” vol. 467, pp. 1–17, 2003.
- [62] M. Golio, *The RF and Microwave Handbook*, ser. Electrical Engineering Handbook. Taylor & Francis, 2010.
- [63] K. Sithamparanathan, M. Marchese, M. Ruggieri, and I. Bisio, *Personal Satellite Services: Second International ICST Conference, PSATS 2010, Rome, Italy, February 4-5, 2010. Revised Selected Papers*, ser. Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering. Springer, 2010.
- [64] Xilinx, Inc., “Xilinx Partial Reconfiguration User Guide (UG702),” pp. 1–130, 2013.
- [65] Mailinglist usrp-users, “Register Reading from the USRP2.” [Online]. Available: <http://thread.gmane.org/gmane.comp.hardware.usrp.e100/2642/>
- [66] F. Schlembach, M. Grimm, and R. S. Thomä, “Real-time Implementation of a DSP-based Algorithm on USRP for Mitigating Non-linear Distortions in the Receiver RF Front-end,” in *The Tenth International Symposium on Wireless Communication Systems 2013 (ISWCS 2013)*, Ilmenau, Germany, Aug. 2013.

# List of Figures

2.1	DCR architecture with the non-ideal components . . . . .	6
2.2	I/Q down-converter using two phase-shifted sinusoids . . . . .	7
2.3	Approximated non-linear input-output characteristic of an amplifier . .	10
2.4	Two-tone test signal applied to the memoryless, polynomial model . . .	12
2.5	Direct form FIR filter . . . . .	13
2.6	FIR filter in the transposed structure . . . . .	14
2.7	AF as an FIR filter with adaptive coefficients . . . . .	15
2.8	AF in the basic configuration for interference cancellation . . . . .	16
2.9	Exemplary FPGA architecture by means of the Xilinx Spartan-3A DSP	23
2.10	FPGA design flow . . . . .	25
3.1	Illustration of the use case scenario . . . . .	28
3.2	Block diagram of the NONLIM algorithm . . . . .	29
4.1	Simplified DSP48A slice model . . . . .	33
4.2	Existing DDC chain with appended NONLIM algorithm . . . . .	36
4.3	Implementation design flow . . . . .	38
4.4	Exemplary HDL module . . . . .	40
4.5	HDL testbench providing stimulus to the HDL module UUT . . . . .	41
4.6	MATLAB-HDL interface interacting with the HDL testbench . . . . .	43
4.7	Custom module being integrated into the DDC chain . . . . .	47
4.8	Filter specifications of a bandstop filter . . . . .	48
4.9	Magnitude and phase response of the quantised FIR filter <i>bsd_filter</i> . .	50
4.10	Simplified diagram of a symmetric systolic FIR filter implementation . .	51
4.11	Generated <i>bsd_filter</i> block to be instantiated within the HDL design . .	53
4.12	Pipelined implementation structure of the <i>refmodel</i> block . . . . .	55
4.13	Implementation structure of the LMS filter . . . . .	58
4.14	LMS filter convergence behaviour . . . . .	62
4.15	HDL simulation with $L = 4, SF = 16, MU = 11$ . . . . .	66

---

5.1	Experimental setup of two-tone scenario . . . . .	71
5.2	Measurement BB spectrum before and after mitigation . . . . .	72
5.3	Experimental setup of BPSK scenario . . . . .	73
5.4	Measurement BB spectrum before and after mitigation . . . . .	74
5.5	Simulation analysis of the adapted reference non-linearity . . . . .	75
6.1	Measurement of BB spectrum of 2 GMSK and a BPSK interferer . . .	78
A.1	Photograph of the experimental setup of the BPSK scenario . . . . .	88

## List of Tables

4.1	Resource utilisation of NONLIM . . . . .	34
4.2	FIR filter specifications with $f_{\text{centre}} = 3 \text{ MHz}$ . . . . .	49
4.3	Resource utilisation of the individual sub-modules of NONLIM . . . . .	67
4.4	Resource utilisation of NONLIM compared to total resources . . . . .	67
4.5	Delays of sub-modules and of NONLIM . . . . .	68
4.6	Utilised DSP48A slices of a real- and complex-valued LMS AF . . . . .	69

# Theses

1. Software-defined radio devices with cheap and simple direct-conversion RF front-end induce RF impairments.
2. Intermodulation products as non-linear distortions (IMDs) can be approximated using a polynomial, memoryless model, which comprises the linear and third-order terms only.
3. The technology of field programmable gate arrays (FPGAs) allows for a flexible and rapid implementation and evaluation of novel signal processing algorithms on real hardware.
4. The non-linearly induced interference mitigation algorithm (NONLIM) is a sophisticated mitigation technique for cleaning a down-converted baseband spectrum of non-linear distortions in real-time.
5. The NONLIM algorithm is successfully implemented on an USRP N210, which is equipped with a Xilinx Spartan-3A DSP FPGA.
6. The IMD products being induced by a two-tone interferer can successfully be mitigated by the NONLIM algorithm.

Ilmenau, 14th of May, 2013

Florian Schlembach



## **Declaration of Authorship**

I, Florian Schlembach, certify hereby that this thesis, entitled “Implementation of DSP-based Algorithms on USRP for Mitigating Non-linear Distortions in the Receiver”, and the work presented in it are, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other university.

Ilmenau, 14th of May, 2013

Florian Schlembach